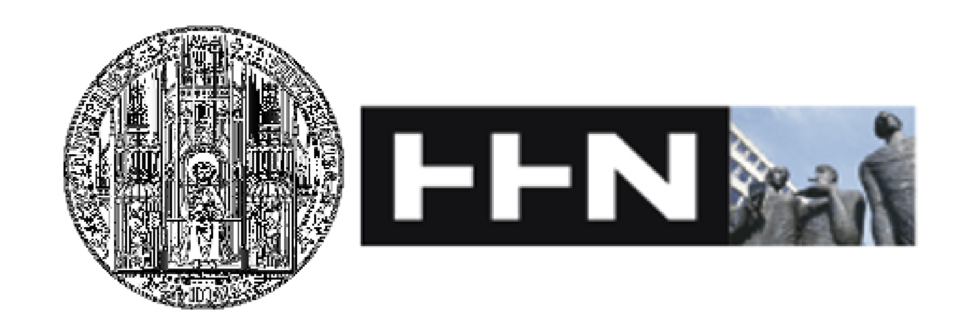


Diplomstudiengang Medizinische Informatik

an der Ruprecht-Karls-Universität Heidelberg, Hochschule Heilbronn

Diplomarbeit

Entwurf und Implementierung der Konfigurations- und Präsentationskomponente eines entscheidungsunterstützenden Systems für die computergestützte Evaluation der kindlichen Spontanmotorik

Diplomand: Eugen Berenstein
Referent: Prof. Dr.-Ing. Hartmut Dickhaus
Korreferent Prof. Dr. Thomas Wetter
Betreuer Dipl. Inf. Med. Dominik Karch
Abgabedatum: 22. April 2011

Abstract

Die visuelle Analyse der kindlichen Spontanmotorik hat sich als aussagekräftiges Verfahren zur Prognose schwerer neurologischer Störungen erwiesen. Eine computergestützte Entscheidungsunterstützung könnte Ärzten eine wertvolle objektive Hilfe bei der Prognosestellung geben.

Mit dem Ziel ein solches entscheidungsunterstützendes System zu entwickeln, wurde am Institut für Medizinische Biometrie und Informatik der Universität Heidelberg ein Forschungsprojekt durchgeführt. Darin wurde u.a. ein Softwarepaket für Forscher entwickelt. Die Ergebnisse der Forschung wurden bislang nicht für Ärzte zugänglich gemacht.

Diese Arbeit ergänzt das Softwarepaket des Projekts um zwei Komponenten.
Ein grafischer Editor erleichtert die Nutzung des bestehenden Softwarepakets, indem es den Forscher bei der Ablaufsteuerung des entscheidungsunterstützenden Prozesses unterstützt. Eine Präsentationskomponente macht die Ergebnisse des entscheidungsunterstützenden Prozesses für Ärzte nutzbar.

Die beiden Komponenten werden mit ähnlichen Komponenten anderer Forschungsprojekte im selben Kontext verglichen und exemplarisch evaluiert.

Dank

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Erstellung der Diplomarbeit unterstützt haben.

Mein Dank gilt Prof. Dr.-Ing. Hartmut Dickhaus, dafür dass ich die Möglichkeit hatte diese Arbeit zu erstellen. Auch Prof. Dr. Thomas Wetter möchte ich für die Übernahme des Korreferats danken.

Herzlich bedanken möchte ich mich bei meinem Betreuer Dipl. Inf. Med. Dominik Karch für seine stetig freundliche Unterstützung, seine Hilfsbereitschaft und Anregungen.

Ganz besonders möchte ich mich bei meiner Familie bedanken, die mich während dieser Zeit immer unterstützt und ermutigt hat.

Inhaltsverzeichnis

1. Einleitung

Die visuelle Analyse der kindlichen Spontanmotorik nach Prechtl [4] ist ein wichtiges Verfahren für die frühe neurologische Untersuchung von Kindern. Es ist schnell und kostengünstig durchführbar und ermöglicht sehr früh eine zuverlässige Prognose schwerer neurologischer Störungen. Der große Nachteil des Verfahrens ist seine Subjektivität. Eine objektive Entscheidungsunterstützung wäre, besonders für unerfahrene Ärzte, eine wertvolle Hilfe.

Es existieren bereits eine Reihe von Arbeiten, die sich dieses Problems angenommen und versucht haben, objektive Parameter aus Bewegungsdaten der Kinder abzuleiten. Auf sie wird in Abschnitt 3.2 eingegangen.

An der Universität Heidelberg wurde ebenfalls ein Forschungsprojekt durchgeführt mit dem Ziel ein computerbasiertes entscheidungsunterstützendes System zu entwickeln, das Ärzten eine objektive Hilfe bei der Analyse der kindlichen Spontanmotorik bieten soll ([13]).

Die vorliegende Arbeit ergänzt das Projekt um zwei Komponenten um diesem Ziel einen Schritt näher zu kommen.

1.1. Gegenstand und Motivation

Rollen im Forschungsprojekt

Das Forschungsprojekt wird in Kooperation mit Kinderärzten und Physiotherapeuten am Universitätsklinikum Heidelberg durchgeführt. Die (Zwischen-)Ergebnisse des Projekts sind dabei für unterschiedliche Benutzergruppen nützlich: *Ärzte* und *Forscher*.

Ärzte sind die Endanwender der Softwarewerkzeuge, die aus den Forschungsergebnissen hervorgehen. Sie wenden die visuelle Analyse der kindlichen Spontanmotorik entweder an oder wollen sie erlernen. Ärzte benötigen objektive Unterstützung bei ihrer Aufgabe.

Forscher untersuchen die kindliche Spontanmotorik und versuchen, mithilfe der Bewegungsparameter[1], Merkmale[2] zu finden, die eine objektive Wertung der Spontanmotorik ermöglichen. Sie nutzen dazu das Wissen der Ärzte über die Qualität der Bewegungen und über auffällige Bewegungsmuster. Forscher benötigen Unterstützung bei der Suche nach Merkmalen in Form von Softwarewerkzeugen.

Aktueller Stand

Im Rahmen des Forschungsprojekts wurden bereits Bewegungsdaten von über 100 Kindern mithilfe eines elektromagnetischen Trackingsystems erfasst. Parallel dazu wurden

[1] Gemeint sind quantitative Parameter der kindlichen Bewegungen.
[2] Quantitative Größen, die aus Bewegungsparametern errechnet werden können.

Videoaufnahmen angelegt. Es wurden bereits einige Merkmale gefunden, die eine Unterscheidung zwischen normalen und abnormalen kindlichen Bewegungen ermöglichen. Es wurde außerdem ein Softwaresystem entwickelt, das das Forschungsprojekt unterstützt.

Darunter wurde auch eine Anwendung entwickelt, die Forscher bei der Extrahierung von Merkmalen aus Bewegungsparametern unterstützt. Die Anwendung wird durch ein Konfigurationscript[3] im XML[4] Format gesteuert. Dieses Script muss vom Forscher unter Einsatz eines XML Editors manuell erstellt werden. Weil XML für den allgemeinen Einsatz entworfen wurde, können übliche XML Editoren keine semantische Unterstützung bieten. Daher ist das manuell erstellte Script fehleranfällig. Weil XML außerdem hierarchisch aufgebaut ist, sind Beziehungen verschiedener Teile des Scripts zueinander nicht offensichtlich. Das führt zu Unübersichtlichkeit von langen Scripten.

Das Softwarepaket wurde konzipiert, um die Forschung in dem Projekt zu unterstützen. Es enthält deshalb keine Komponente, die die Forschungsergebnisse für Ärzte nutzbar macht, indem sie gefundene Merkmale dem Arzt in angemessener Form präsentiert.

In Abschnitt 2.2 wird das Softwarepaket näher beschrieben. In Abschnitt 2.2 „Berechnung von Merkmalen“ auf Seite 14 wird näher auf den bisherigen Ablauf der Extrahierung von Merkmalen eingegangen.

Gewünschte Ergänzungen

Für die weitere Suche nach Merkmalen ist eine Konfigurationskomponente erwünscht, die das bestehende Softwaresystem ergänzt und die Konfiguration der Anwendung zur Extrahierung von Merkmalen erleichtert. Sie soll im Gegensatz zum bisherigen Verfahren dem Forscher semantische Unterstützung und eine bessere Übersichtlichkeit bieten und dadurch die Fehlerrate bei der Konfiguration senken.

Als Schritt in Richtung entscheidungsunterstützendes System wird außerdem ein Ansatz für eine Visualisierungskomponente benötigt. Diese soll gefundene Merkmale und Bewegungsparameter für den beurteilenden Arzt visualisieren und eine Möglichkeit bieten, die unterschiedlichen Fälle anhand der Merkmale miteinander vergleichen zu können. Dadurch soll der Arzt bei der Beurteilung der kindlichen Spontanmotorik sowie beim Erlernen der Beurteilungstechnik unterstützt werden.

1.2. Zielsetzungen

Die vorliegende Arbeit verfolgt zwei Ziele: Die Entwicklung einer *Konfigurationskomponente* für den Forscher und die Entwicklung eines Ansatzes für eine *Visualisierungskomponente* für den Arzt.

[3] Enthält eine Reihe von Anweisungen, die die Einstellungen einer Anwendung setzen.
[4] Weit verbreitete Sprache zur Darstellung hierarchisch strukturierter Daten in Textform.

Die zu entwickelnde *Konfigurationskomponente* soll dem Forscher die Konfiguration der Anwendung zur Extrahierung von Merkmalen erleichtern. Sie soll semantische Unterstützung und bessere Übersichtlichkeit bei der Erstellung des Konfigurationscripts bieten, indem sie:

- Die Konfiguration grafisch repräsentiert (und grafisch editierbar macht).
- Das Konfigurationscript validiert.
- Bestimme Aufgaben automatisiert (z.B. Teile des Scripts automatisch anlegt).

Der Ansatz für eine *Visualisierungskomponente* soll Ärzte bei der Beurteilung der Spontanmotorik durch objektive Werte unterstützen, indem die Visualisierungskomponente:

- Merkmale zusammen mit Bewegungsparametern und Videoaufnahmen in übersichtlicher Form visualisiert.
- Die Möglichkeit bietet, unterschiedliche Fälle anhand der Merkmalswerte zu vergleichen.
- Die Möglichkeit bietet, markierte Segmente in der Aufnahme einzeln abzuspielen.

Die Komponente soll erweiterbar sein, so dass neue Merkmale einfach hinzugefügt werden können. Die Komponente soll außerdem bezüglich der Daten, die visualisiert werden sollen, konfigurierbar sein. Das soll es dem Forscher ermöglichen, den Ärzten dynamisch einen sinnvollen Umfang an visualisierten Daten zur Verfügung zu stellen.

Die Konfigurationskomponente soll evaluiert werden, indem ein Vergleich des bisherigen Verfahrens mit dem Einsatz der Komponente anhand einer realen Konfiguration durchgeführt wird.

Die Visualisierungskomponente soll zur Evaluierung Ärzten des Universitätsklinikums Heidelberg, die mit dem Forschungsprojekt kooperieren, vorgestellt und auf die Tauglichkeit des Ansatzes hin, die Arbeit der Ärzte zu unterstützen beurteilt werden.

1.3. Gliederung

Es folgt eine kurze Beschreibung des Inhalts der einzelnen Kapitel dieser Arbeit.

Kapitel 2: „Grundlagen" gibt eine Einführung in das Gebiet in dem sich diese Arbeit aufhält. Es wird die visuelle Beurteilung der kindlichen Spontanmotorik erläutert sowie ihre Problematik. Es wird auch erläutert, wie das bisher entwickelte Softwarepaket des Forschungsprojekts, den Forscher unterstützt und die Problematik bei seinem Einsatz.

Kapitel 3: „Analyse bestehender Ansätze" analysiert den Ansatz von grafischen Editoren anhand eines Beispels. Es werden auch verschiedene bestehende Ansätze für die Visualisierung der kindlichen Bewegungsdaten als Entscheidungshilfe für Ärzte vorgestellt und mit dem hier vorgestellten Ansatz verglichen.

Kapitel 4: „Konfigurationskomponente" stellt die Konzeption der Konfigurationskomponente und ihre Realiserung vor. Am Ende des Kapitels wird die Komponente evaluiert.

Kapitel 5: „Visualisierungskomponente" stellt die Konzeption der Visualisierungskomponente und ihre Realisierung vor. Am Ende des Kapitels wird die Komponente evaluiert.

In Kapitel 6: „Diskussion und Ausblick" werden die beiden Komponenten kritisch betrachtet und ein Ausblick für die weitere Entwicklung gegeben.

2. Grundlagen

In diesem Kapitel wird die visuelle Beurteilung der kindlichen Spontanmotorik erläutert sowie ihre Problematik. Es wird auch erläutert, wie das bisher entwickelte Softwarepaket des Forschungsprojekts, den Forscher unterstützt und die Problematik bei seinem Einsatz.

2.1. Beurteilung der kindlichen Spontanmotorik

Das kindliche Nervensystem generiert spontan viele verschiedene Bewegungsmuster. Eine Untergruppe dieser Muster sind die *General Movements*. Nach Prechtl [20] sind dies grobe Bewegungen, die den ganzen Körper mit einbeziehen. Sie bestehen aus einer variablen Sequenz von Arm-, Bein-, Hals und Rumpfbewegungen. Sie nehmen an Intensität, Kraft und Schnelligkeit abwechselnd ab und zu und bauen sich langsam auf oder ab. Drehbewegungen um die Extremitätenachsen und leichte Änderungen der Bewegungsrichtung lassen die Bewegungen flüssig und elegant aussehen und erzeugen den Eindruck von Komplexität und Variabilität.
General Movements werden in drei Phasen entsprechend dem Alter des Kindes mit unterschiedlichen Charakteristika der Bewegungen eingeteilt (siehe [10, Table 1]): ***Preterm* GMs**, beim Fötus und neugeborenen Kind; ***Writhing* GMs**, um den Zeitpunkt des Geburtstermins und während der ersten zwei Lebensmonate nach dem Geburtstermin; ***Fidgety* GMs**, ab der 9. bis 10. Lebenswoche bis zu ca. 5 Monate nach dem Geburtstermin. Danach werden die *General Movements* zunehmend von willkürlichen Bewegungen abgelöst. Dabei haben diese Phasen fließende Grenzen und können sich sogar zeitweise überlappen (Abbildung 2.1 veranschaulicht das).

Wegen ihrer Komplexität, ihres häufigen Auftretens und langer Dauer eignen sich *General Movements* zur genauen Beobachtung. Verschiedene Studien ([5, 21, 9]) haben gezeigt, dass sich die Qualität der *General Movements* bei Gehirnläsionen signifikant ändert.

Hadders-Algra teilt die Qualität der *General Movements* in vier Klassen ein: Normal-optimal, Normal-suboptimal, Mildly abnormal und Definitely abnormal ([10]). Die Klassenzugehörigkeit wird anhand der drei Hauptparameter der GM Qualität bestimmt: Komplexität (räumliche Variabilität der Bewegungen), Variation (zeitliche Variabilität der Bewegungen) und Flüssigkeit der Bewegungen (allmähliche Zu- und Abnahme der Geschwindigkeit).

Die Parameter der Bewegungsqualität werden in einem standardisierten Verfahren von ausgebildeten Ärzten visuell beurteilt.

Für eine korrekte Beurteilung müssen Ärzte speziell ausgebildet werden. Einspieler nennt 83% korrekter Beurteilungen nach einer Basisausbildung und eine signifikant höhere Rate von 88% nach einer Zusatzausbildung [4, Abschnitt: Training Required]. Die Ausbildungskurse dauern vier bis fünf Tage.

Beurteilungsprozess

Die Beurteilung der *General Movements* wurde am Universitätsklinikum Heidelberg in einen klinischen Prozess integriert. Bevor der Prozess erklärt wird, müssen einige klinische Begriffe eingeführt werden.

Befund Der Befund betrifft die *General Movements* Qualität des Kindes. Sie wird in zwei Klassen eingeteilt:

- *DA* steht für definitely abnormal GMs.
- *NO* darunter werden die anderen drei Qualitäten der GMs zusammengefasst: mildly abnormal, normal-suboptimal und normal-optimal.

Outcome Der Outcome bezeichnet die endgültige Diagnose bezüglich der neuronalen Störung des Kindes. Er wird in zwei Klassen eingeteilt:

- *ICP* steht für Infantile Cerebralparese, dieser Begriff bezeichnet eine Störung des Muskel- und Nervensystems, hervorgerufen durch eine frühkindliche Hirnschädigung.
- *kein ICP* bezeichnet gesunde Kinder.

Im Folgenden wird der Prozess beschrieben, den der Arzt bei der Beurteilung der kindlichen Spontanmotorik durchläuft. Dieser Prozess wird vorzugsweise während allen drei GM Phasen des Kindes wiederholt. Das ermöglicht die Konsistenz der GM Befunde zu überprüfen und wird von Prechtl empfohlen [20].

1. Es wird eine Aufnahme des Kindes entsprechend den Kriterien der GM Analyse ([4]) angefertigt (sowohl eine Videoaufnahme als auch eine Aufnahme der Bewegungsdaten). Bei der Aufnahme muss der beurteilende Arzt nicht anwesend sein.
2. Der beurteilende Arzt studiert die Videoaufnahme und erstellt anhand der GM Qualitätskriterien einen Befund.
3. Wenn der Befund DA lautet, steht das Kind unter Risiko zur Entwicklung von ICP. Dann wird eine physiotherapeutische Behandlung eingeleitet.

Im Alter von 2 Jahren, wird eine Nachuntersuchung des Kindes durchgeführt. Dabei wird bei einer neurologischen Untersuchung der Outcome festgestellt.

Schwachstellen

Obwohl das Verfahren eine frühe und zuverlässige Prognose für spätere neurologische Entwicklungsstörungen erlaubt, das außerdem schnell durchführbar und kostengünstig ist, ist es subjektiver Natur. Die Hauptparameter der GM Qualität: Variation, Komplexität und Flüssigkeit der Bewegungen sind subjektive Begriffe und lassen sich nicht in objektiven Werten ausdrücken. Ärzte brauchen eine spezielle Ausbildung und viel Erfahrung, um das Verfahren effektiv anwenden zu können. Besonders noch unerfahrene Ärzte haben

deutlich weniger korrekte Auswertungen ([4]). Es fehlt an objektiven Entscheidungshilfen für die Ärzte.

Auch der Vergleich unterschiedlicher Fälle miteinander kann nur auf subjektiver Basis durchgeführt werden. Ärzte können nur Fälle vergleichen, die sie kennen und für vergleichbar halten. Es fehlt also auch an objektiven Vergleichskriterien für GM Aufnahmen.

2.2. Das Softwarepaket des Forschungsprojekts

Im Rahmen des Forschungsprojekts wurde ein Softwarepaket entwickelt, das die Erfassung, Verarbeitung und Analyse von kindlicher Spontanmotorik ermöglicht.

Es ist in Komponenten unterteilt, die miteinander über standardisierte Schnittstellen kommunizieren. Diese Schnittstellen können auch genutzt werden, um mit anderen Softwareprodukten auf die Daten zuzugreifen.

Es enthält drei Datenbanken: Videodatenbank, Datenbank für klinische Daten und Datenbank für Bewegungsdaten. Die **Videodatenbank** enthält Videoaufnahmen der Kinder für die GM Beurteilung. Die Datenbank für **klinische Daten** enthält Stammdaten der Kinder, Diagnosen und Befunde. Die Datenbank für **Bewegungsdaten** enthält die Bewegungsparameter und die berechneten Merkmale für jede Aufnahme.

Außerdem enthält es verschiedene Softwarekomponenten. Die Komponente **Aufzeichnung** ermöglicht die Aufnahme der Videos und Bewegungsdaten. Die Komponente **Modellberechnung** berechnet Bewegungsparameter aus den aufgezeichneten Bewegungsdaten. Die Komponente **Annotation**, ermöglicht es Ärzten, Videoaufnahmen anzuschauen und bezüglich der GM Qualität zu annotieren. Die Komponente **Visualisierung** ermöglicht es Bewegungsdaten dreidimensional zu visualisieren. Die Komponente **Datenverarbeitung und -Analyse** ist dafür zuständig, neue Zeitreihen und Merkmale aus den Bewegungsdaten zu gewinnen, die eine Unterscheidung bezüglich der GM Qualität ermöglichen.
Abbildung 2.2 veranschaulicht die Kommunikation der Komponenten des Softwarepakets.

Die Daten der Datenbank für **Bewegungsdaten** sind so organisiert, dass es pro Aufnahme mehrere Teilaufnahmen gibt, pro Teilaufnahme wiederum mehrere *Zeitreihen*, *Segmentlisten* und *Merkmale* (siehe Abbildung 2.3).

Zeitreihen beschreiben Bewegungsparameter der Bewegungsaufnahme. Beispielsweise den Winkel zwischen zwei Körpersegmenten zu jedem Zeitpunkt der Aufnahme. Sie werden von den Komponenten **Modellberechnung** und **Datenverarbeitung und -Analyse** erzeugt.

Segmentlisten beschreiben Zeitausschnitte der Aufnahme, die bestimmte Eigenschaften aufweisen (zu vergleichen mit einer Videoschnittliste).

Merkmale werden von der Komponente **Datenverarbeitung und -Analyse** generiert. Sie beschreiben bestimmte Qualitätsaspekte der Bewegungen oder können für die Klassifikation der Bewegungsdaten verwendet werden.

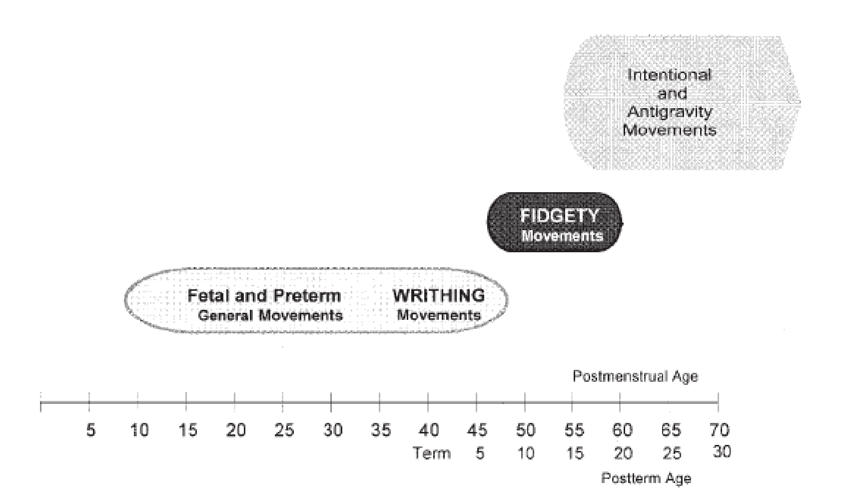


Abbildung 2.1: Zeitliche Einteilung der Phasen mit unterschiedlichen Bewegungscharakteristika bei General Movements. Aus [4]

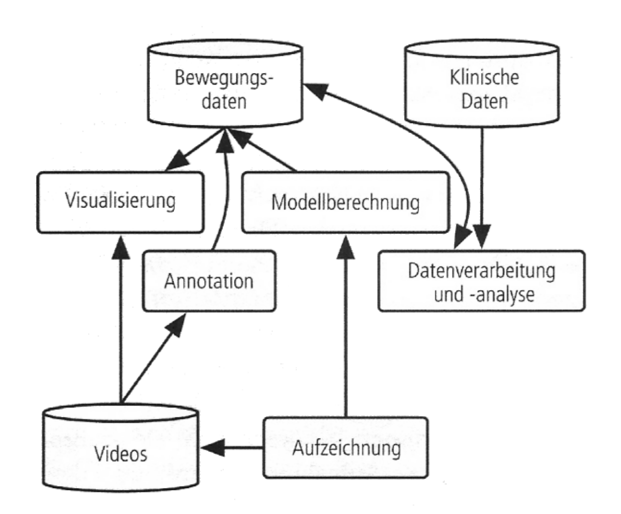


Abbildung 2.2: Komponenten des Softwarepakets im Forschungsprojekt und ihre Kommunikation untereinander. (aus [13])
Runde Formen symbolisieren Datenbanken.
Rechteckige Formen symbolisieren Softwarekomponenten.
Pfeile symbolisieren Datenflüsse (in Richtung der Pfeilspitze).

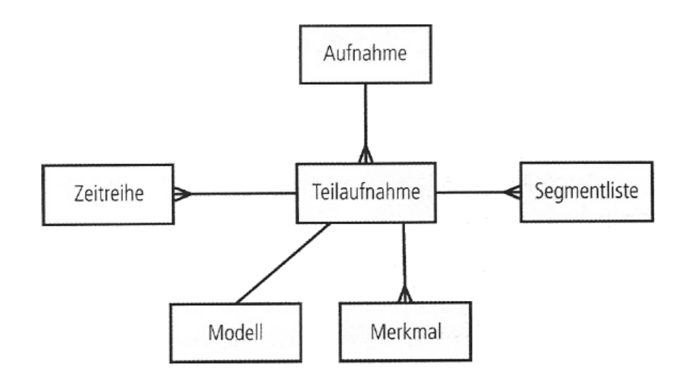


Abbildung 2.3: Relevanter Ausschnitt des Datenmodells der Datenbank für Bewegungsdaten (aus [13]).
Rechtecke symbolisieren Elemente des Datenmodells.
Linien zwischen ihnen symbolisieren Beziehungen.
Aufgefächerte Linienenden bedeuten, dass an diesem Ende der Beziehung mehrere Elemente sein können.

Berechnung von Merkmalen

Merkmale werden von der Komponente **Datenverarbeitung und -Analyse** berechnet. Sie wird vom Forscher eingesetzt um neue Informationen aus den Bewegungsdaten zu gewinnen, die eine Unterscheidung bezüglich der GM Qualität ermöglichen.

Anwendungsstruktur

Die Komponente besteht aus drei Teilen: dem *Konfigurationscript*, den *Algorithmen* und dem *Scriptausführungsmodul*. Abbildung 2.4 stellt den Entwurf der Komponente schematisch dar.

Algorithmen Sind in diesem Kontext Programmmodule zur Berechnung von bestimmten Ausgangswerten aus Eingangswerten. Sie haben eine standardisierte Schnittstelle, Ein- und Ausgänge und Parameter.

Konfigurationscript Beschreibt die Berechnungsvorschrift für ein Merkmal durch Definition einer Sequenz von Algorithmen, Verknüpfung ihrer Eingänge mit Daten und ihrer Ein- und Ausgänge untereinander.

Scriptausführungsmodul Setzt die Berechnung um, die durch das Konfigurationscript definiert wurde, indem es erforderliche Daten lädt, die Algorithmen mit Daten versorgt und ausführt und Ergebnisse in der Datenbank abspeichert.

Eine solche Trennung der Komponente hat mehrere Vorteile:

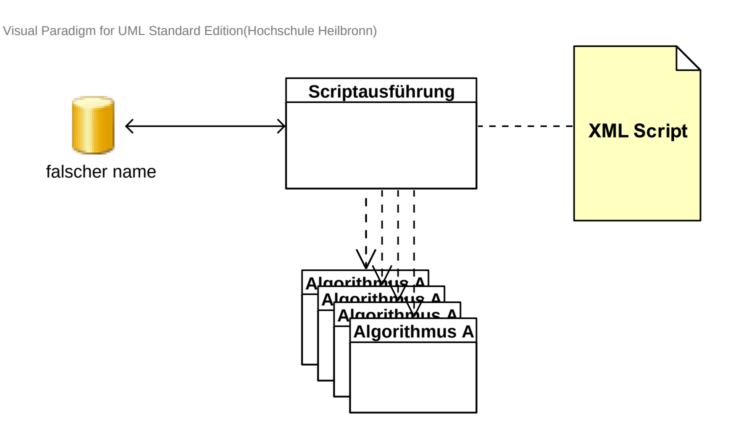


Abbildung 2.4: Entwurf der Anwendung zur Berechnung von Merkmalen (Schematisch).

- Algorithmen können unabhängig von ihrer Versorgung mit Daten implementiert werden. Daten können dem Algorithmus flexibel aus unterschiedlichen Quellen zur Verfügung gestellt werden. Neue Algorithmen können einfach hinzugefügt werden.
- Der Forscher hat die Möglichkeit, Algorithmen beliebig zu kombinieren, um die benötigten Berechnungen auszuführen. Die Kombination erfolgt deklarativ[5], das Programm muss dazu nicht geändert werden.

Damit dieser Ansatz funktioniert muss das *Scriptausführungsmodul* die Daten von und zu den Algorithmen in generischer Form kommunizieren. Das geschieht in Form einer Auflistung von Objekten, auf die über ihren Namen zugegriffen werden kann. Die Namen werden durch das Konfigurationscript festgelegt. Ein Algorithmus benötigt bestimmte Daten und erwartet sie, unter einem fest definierten Namen und mit einem definierten Datentyp[6], in der Auflistung zu finden.

Listing 1: Beispiel des Datenzugriffs eines Algorithmus mit Typumwandlung

```
var data = (Datatype)InputData["dataname"];
```

Arbeitsablauf

Abbildung 2.5 stellt schematisch den Arbeitsablauf dar, den die Komponente Datenverarbeitung und -Analyse durchläuft, wenn ein Merkmal berechnet wird. Dabei ist anzumerken, dass die Scriptausführungskomponente kein Kompilierer[7] ist und es sich bei

[5]D.h. der Anwender definiert was getan werden soll, nicht wie es getan wird.

[6]Gemeint sind bestimmte Arten von Objekten mit darauf definierten Operationen bei typisierten Programmiersprachen.

[7]Kompilierer (Compiler) und Kompilieren sind Begriffe aus der Informatik, die das Übersetzen von Programmen aus einer Sprache in eine andere Sprache bezeichnen.

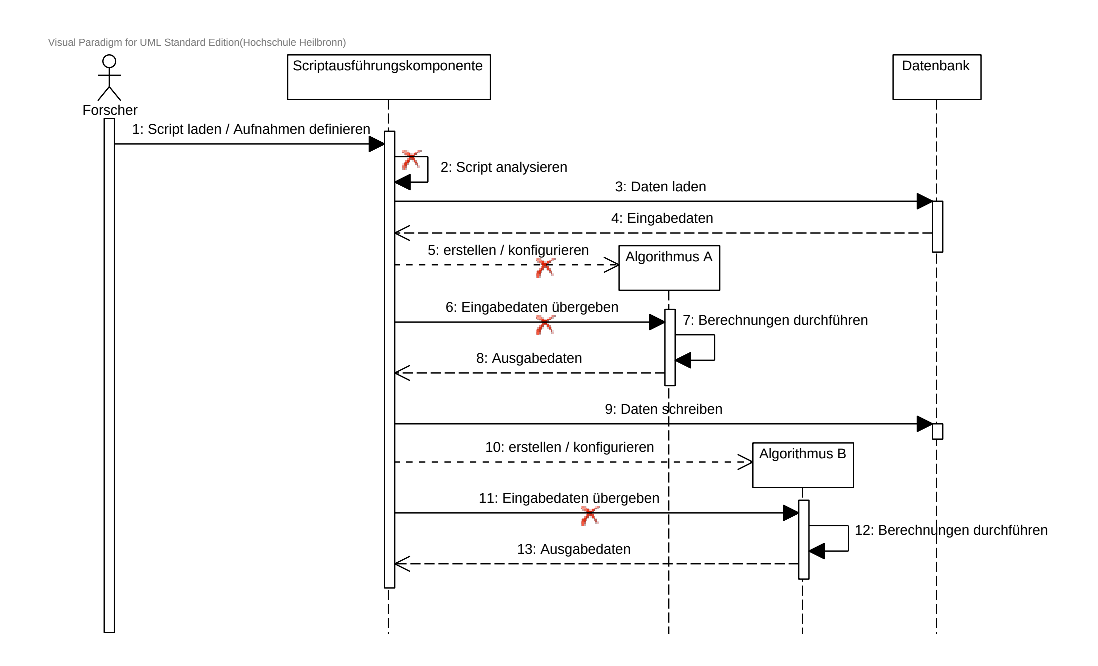


Abbildung 2.5: Arbeitsablauf der Komponente Datenverarbeitung und -Analyse bei der Berechnung von Merkmalen. Sequenzdiagramm.

diesem Ablauf nicht um einen Kompiliervorgang handelt, sondern um eine Interpretation des Scripts und Abarbeitung der darin definierten Schritte. Die Schritte in der folgenden Auflistung entsprechen den Nummern in der Abbildung.

Schritt 1 Der Forscher lädt ein Konfigurationscript in der Scriptausführungskomponente und definiert die Aufnahmen, auf die das Script angewendet werden soll.

Schritt 2 Die Komponente liest das Script ein und analysiert es.

Schritt 3 Die Komponente lädt Eingabedaten für den ersten auszuführenden Algorithmus aus der Datenbank.

Schritt 5 Die Komponente erstellt den ersten Algorithmus und konfiguriert seine Parameter.

Schritt 6 Die Komponente übergibt dem Algorithmus seine Eingabedaten.

Schritt 7 Der Algorithmus führt Berechnungen durch und gibt Ausgabedaten zurück.

Schritt 9 Die Komponente schreibt eventuell Daten in die Datenbank.

Schritt 10 Die Komponente erstellt und konfiguriert den zweiten Algorithmus.

Schritt 11 Die Komponente übergibt dem Algorithmus seine Eingabedaten. Dabei können die Daten sowohl aus der Datenbank stammen, als auch aus der Ausgabe des ersten Algorithmus (das wird durch das Script festgelegt).

Schritt 12 Der Algorithmus führt Berechnungen durch und gibt Ausgabedaten zurück.

Dieser Ablauf wird fortgesetzt, bis das Konfigurationscript vollständig abgearbeitet wurde. Dabei können an mehreren Stellen Fehler auftreten, die auf eine falsche Angabe im Script zurückzuführen sind. Im folgenden werden einige der Fehler aufgelistet.

In *Schritt 2* kann ein Fehler auftreten, wenn das Script syntaktisch nicht korrekt ist. Um ihn zu vermeiden muss der Forscher die Syntax des Konfigurationscripts genau kennen.

In *Schritt 5* kann durch inkorrekte Angabe des vollständigen Namens des Algorithmus ein Fehler bei der Erstellung auftreten. Bei der Konfiguration des Algorithmus kann es zu einem Fehler kommen wenn Parameter falsch angegeben oder mit falschen Werten besetzt werden. Dieser Fehler kann vermieden werden, wenn der Forscher eine genaue Kenntnis über die Definition des Algorithmus und seiner Parameter verfügt.

In *Schritt 6* kann ein Fehler auftreten, wenn dem Algorithmus unzureichende Daten, Daten falschen Datentyps oder unter falschen Namen übergeben werden. Dieser Fehler kann durch genaue Kenntnis über die Implementierung des Algorithmus vermieden werden.

Zusätzlich zu den Ursachen in Schritt 6 kann in *Schritt 11* ein Fehler auftreten, wenn der Datentyp eines Ausgabedatums des ersten Algorithmus nicht mit dem Datentyp übereinstimmt, den der zweite Algorithmus erwartet. Die Vermeidung dieses Fehlers erfordert dass der Forscher die Implementierung beider Algorithmen im Blick hat, wenn er sie zusammensetzt.

Problematik

Da es den üblichen XML Editoren an semantischer Unterstützung fehlt, kann der Forscher beim Erstellen des Konfigurationscripts leicht Fehler machen. Die Fehler können darin bestehen dass er falsche Namen für Algorithmen angibt, die Datentypen der angegebenen Daten nicht übereinstimmen, die Eingabedaten des Algorithmus falsch definiert werden etc.

Durch den generischen Aufbau der Anwendung äußern sich diese Fehler erst zur Ausführungszeit des Scripts. Die Fehlersuche im Script ist dabei, besonders bei langen Scripten, nicht einfach.

3. Analyse bestehender Ansätze

Dieses Kapitel analysiert den Ansatz von grafischen Editoren anhand eines Beispels. Es werden auch verschiedene bestehende Ansätze für die Visualisierung der kindlichen Bewegungsdaten als Entscheidungshilfe für Ärzte vorgestellt und mit dem hier vorgestellten Ansatz verglichen.

3.1. Konfiguration durch grafische Editoren

Eigenschaften grafischer Editoren

Grafische Editoren abstrahieren von Daten oder einem System. Sie ermöglichen dem Benutzer mit Daten oder Systemen zu arbeiten, deren Struktur und Syntax er nicht kennt. Sie bieten eine intuitive Benutzung ohne großen Lernaufwand. Durch die grafische Repräsentation kann der Benutzer eine bessere Übersicht über die Daten oder das System gewinnen.

Grafische Editoren haben alledings eine höhere Komplexität und daher auch eine höhere Fehleranfälligkeit verglichen mit textbasierten Editoren. Die höhere Komplexität führt zu einem höheren Entwicklungsaufwand. Sie erfordern die explizite Implementierung eines Datenmodells. Sie erfordern auch die Implementierung der grafischen Repräsentation des Modells und Werkzeugen, mit denen die Repräsentation manipuliert werden kann. Die Erweiterung des Editor um neue Komponenten ist aufwändig, da sowohl das Modell als auch die grafische Repräsentation entwickelt und die Funktionalität des Editors an die neue Komponente angepasst werden muss.

Der Einsatz von speziellen Frameworks kann die Entwicklung und Erweiterung grafischer Editoren erleichtern.

Anwendungen grafischer Editoren

Als Beispiel wird eine Arbeit von Huber ([12]) angeführt, in der ein grafischer Editor zur Erstellung von Zellmodellen für eine Simulationssoftware entwickelt wurde.

Den Rahmen für diese Arbeit bildete ein Projekt, bei dem eine Software zur Beschreibung und Simulation des Verhaltens von Haut am Computer entwickelt wurde. Dabei wurden einzelne Zellen als Automaten modelliert, die miteinander interagieren. Die Modelle der Zellen wurden als Programmcode erstellt, der fest in die Simulationssoftware integriert war.

Huber entwickelte einen grafischen Editor, der die Modellierung von Zellen ermöglichte. Aus dem Modell wurde Programmcode erzeugt, der in die Simulationssoftware integriert werden konnte. Dies erlaubte die Erstellung von Zellmodellen, ohne Kenntnisse über die Implementierung der Modelle und ihre Integration zu benötigen. Außerdem bot der

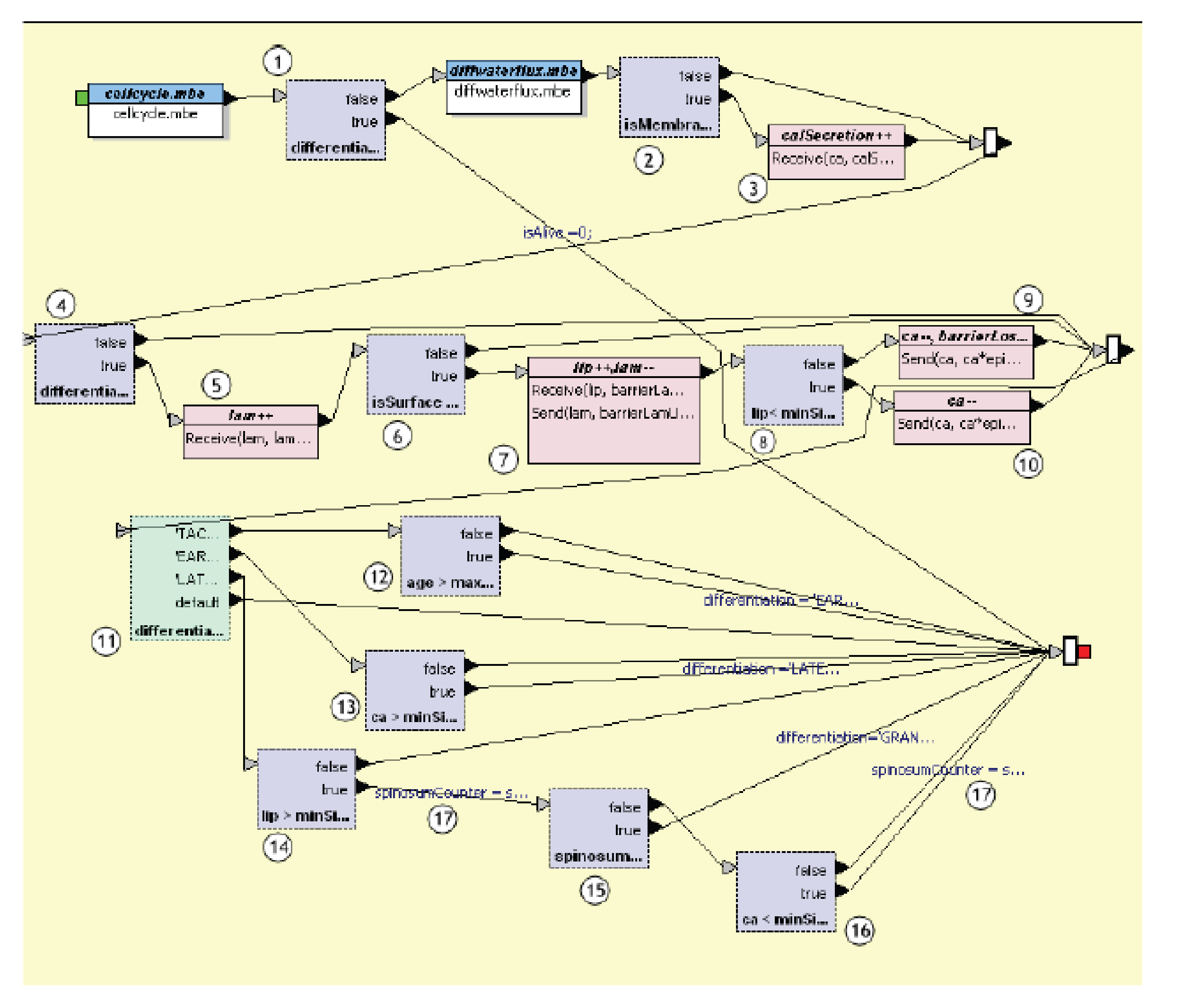


Abbildung 3.1: Modell eines Zellzyklus mit dem grafischen Editor von Huber. Aus [12].

Editor Validierung der Modelle an. Abbildung 3.1 zeigt das Modell eines Zellzyklus, das mit dem grafischen Editor erstellt wurde.

Der Editor wurde auf Basis von Eclipse unter Einsatz eines Frameworks für grafische Editoren erstellt.

Dieser Editor bietet einer breiten Benutzergruppe die Möglichkeit, dynamisch komplexe Zellmodelle zu erstellen.

Fazit

Der Ansatz, die Konfigurationskomponente als grafischen Editor zu realisieren scheint erfolgsversprechend. Dafür sprechen die Vorteile grafischer Editoren. Mithilfe eines Frameworks zur Erstellung grafischer Editoren können die Schwierigkeiten bei dessen Entwicklung bewältigt werden. Der Erfolgreiche Einsatz des grafischen Editor bei *Huber* bestätigt dies.

3.2. Visualisierung von Bewegungsparametern

Vorstellung verwandter Projekte

Eine Reihe anderer Projekte als das zugrundeliegende Forschungsprojekt hat sich mit derselben Problematik befasst. Die Zielsetzung aller Projekte auf diesem Gebiet ist es, objektive Parameter aus den kindlichen Bewegungen zu extrahieren, die eine Unterscheidung der Bewegungsqualität ermöglichen und so die visuelle Analyse der Spontanmotorik unterstützen.

Adde et al ([1]) konzentrieren sich auf eine Untergruppe der General Movements, die Fidgety Movements (siehe [4]). Sie nutzen computergestützte Videoanalyse, um das Vorhandensein oder Nichtvorhandensein von FMs zu erkennen.

Sie präsentieren eine Software, die aus Videoaufnahmen Bewegungsbilder extrahiert. Aus den Bewegungsbildern berechnet die Software zwei grafische Darstellungen. Der Bewegungsgraph zeigt die Anteile der Bewegung in jeder Körperregion über die Zeit. Der Bewegungszentroid zeigt die Bewegung des Bewegungszentrums über die Zeit. Aus diesen Graphen werden quantitative Parameter berechnet: der Anteil an Bewegung über die Zeit und die Standardabweichung der Bewegung des Zentrums. Diese Parameter werden zur Klassifikation verwendet.

Meinecke et al ([16]) modellieren subjektive Parameter, die von Ärzten zur Beurteilung eingesetzt werden, als matematische und statistische Parameter. Sie nutzen ein optisches Trackingsystem, um Bewegungsdaten aufzunehmen und klassifizieren sie anhand der berechneten Parameter.

Zum Beispiel wird der subjektive Parameter *Flüssige Bewegungen* von Meinecke el al wie folgt modelliert: Fläche, in der der Bewegungsverlauf außerhalb seines gleitenden

Mittelwerts und seiner Standardabwechung ist.
Aus den Parametern wurde mithilfe eines mathematischen Optimierungsverfahrens eine optimale Kombination von wenigen Parametern ausgewählt.
Sie präsentieren außerdem eine Software, die anhand der berechneten Merkmale eine Klassifikation der Aufnahmen durchführt.

Berge et al ([2]) haben eine Software entwickelt, die bei der Modellierung von objektiven Merkmalen unterstützten soll. Dazu nutzen sie Visualisierungen von Bewegungsdaten, die einem bestimmten Bewegungsmuster entsprechen und versuchen aus ihnen Merkmale zu extrahieren.

Die Software präsentiert dem Benutzer verschiedene Visualisierungen bestimmter Bewegungsmuster. Für die Visualisierung werden allgemeine Verfahren wie die Hauptachsentransformation und eine Zeit-Frequenz Darstellung eingesetzt. Die Visualisierungen werden so optimiert, dass sie mit dem Bewegungsmuster korrelieren, dann werden aus ihnen objektive Merkmale extrahiert, die das Bewegungsmuster repräsentieren.

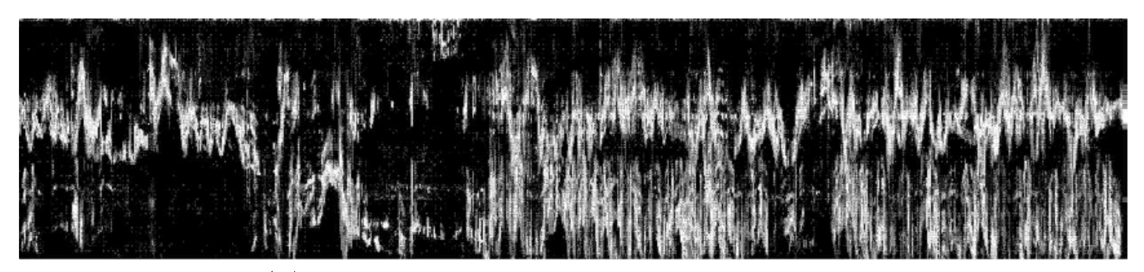
Ansatz dieser Arbeit

Im Folgenden wird beschrieben, wie das zugrundeliegende Forschungsprojekt seine Zielsetzung realisiert. Ausführliche Erklärungen und Beispiele findet der Leser in [13].

Der Forscher beobachtet die Bewegungen der Kinder unter Einbeziehung der errechneten Bewegungsparameter und versucht Auffälligkeiten festzustellen, die Kinder mit Outcome *ICP* oder Befund *DA* aufweisen. Dabei arbeitet er mit Ärzten zusammen. Als nächstes formalisiert er die gefundenen Auffälligkeit als Merkmal, d.h. er modelliert sie durch einen Algorithmus. Er berechnet dieses Merkmal für mehrere Kinder und untersucht, ob es tatsächlich mit dem Outcome oder dem Befund korreliert ist.

Die Berechnung des Merkmals, die automatisch durchgeführt werden kann, generiert unterschiedliche Daten. Der Wert des Merkmals gibt eine zusammenfassende Bewertung eines bestimmten Bewegungsmusters für die gesamte Aufnahme. Es werden Segmentlisten erzeugt, die die Stellen in der Aufnahme, an denen das Bewegungsmuster auffällt, markieren. Es werden Zeitreihen erzeugt, die die Ausprägung des Merkmals über die Zeit beschreiben.

Diese Daten werden nach der Integration des Merkmals in die Visualisierungskomponente den Ärzten präsentiert. Der Merkmalswert kann als Vergleichskriterium zwischen unterschiedlichen Fällen genutzt werden und fließt in die Klassifikation der Gesamtaufnahme ein. Segmente können in der Videoaufnahme angesprungen und abgespielt werden. Dies ermöglicht auch den Vergleich derselben Auffälligkeiten bei unterschiedlichen Kindern. Die Zeitreihen des Merkmals erlauben eine detailierte Analyse und geben einen Einblick wie der Algorithmus verschiedene Bewegungsmuster bewertet.



(a) Bewegungsgraph eines Kindes mit Fidgety Movements.

(b) Bewegungsgraph eines Kindes, das keine Fidgety Movements aufweist.

Abbildung 3.2: Bewegungsgraphen von zwei Aufnahmen aus [1]. Die x-Achse ist die Zeit. Die y-Achse vertikal die Stelle der Bewegung im Körper.

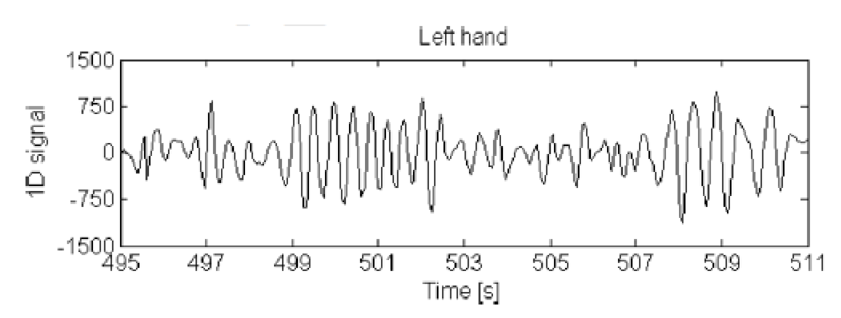
Vergleich der Ansätze

Der Ansatz von *Adde et al* führt eine Reduktion der Videoaufnahme auf eine abstrakte Darstellung durch, die Bewegungsinformationen des ganzen Körpers kumuliert. Diese Darstellung wird dem Benutzer zusammen mit dem Video präsentiert. Die Berechnung der Darstellung und der quantitativen Parameter erfolgt semiautomatisch. Der Benutzer muss zuvor die interessante Region im Video auswählen. Abbildung 3.2 zeigt Beispiele von Bewegungsgraphen.

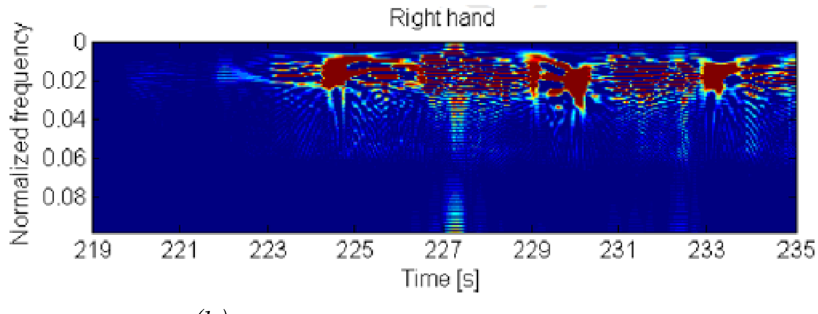
Es ist fraglich, ob die abstrakte Darstellung einen Mehrwert für Ärzte bei der Beurteilung der GM bietet. Es ist auch fraglich, ob die Klassifizierung nach dem Vorhandensein oder Nichtvorhandensein von Fidgety Movements für eine verlässliche Bewertung ausreicht, da Prechtl ([20]) viele weitere Aspekte der GM nennt und eine Beobachtung über alle drei Phasen der GM empfiehlt.

Der Ansatz von *Meinecke et al* basiert auf Modellen der subjektiven Parameter durch statistische und mathematische Parameter. Eine optimale Kombination dieser Parameter wird automatisch ausgewählt und die Parameter berechnet. Dem Benutzer werden Bewegungsparameter in verschiedenen Darstellungen und das Klassifikationsergebnis ahand der errechneten Parameter präsentiert.

Die präsentierten Bewegungsparameter allein ohne Bezug zur Videoaufnahme sind für den Arzt schlecht nutzbar. Die ausgiebige Nutzung von statistischen und mathematischen Methoden, sowohl bei der Berechnung als auch bei der Bewertung der objektiven



(a) Ergebnis der Hauptachsentransformation angewendet auf Bewegungsparameter.



(b) Wagner-Ville Zeit-Frequenz Darstellung.

Abbildung 3.3: Verschiedene Visualisierungen von Bewegungsmustern aus [2].

Parameter machts es den Ärzten sehr schwer die Entscheidungen der Klassifikationssoftware nachzuvollziehen. Es ist außerdem fraglich, ob die Modellierung der subjektiven Parameter durch allgemeine statistische und mathematische Methoden ausreichend ist um die große Variabilität der kindlichen Bewegungen abzudecken.

Der Ansatz von *Berge et al* unterscheidet sich von den anderen beschriebenen Ansätzen. Die anderen Ansätze (und auch der Ansatz dieser Arbeit) präsentieren dem Benutzer bereits modellierte quantitative Parameter. Dieser Ansatz präsentiert dem Benutzer dagegen anpassbare visuelle Modelle von auffälligen Bewegungsmustern und versucht aus ihnen passende objektive Parameter zu extrahieren. Dem Benutzer wird außerdem die Videoaufnahme präsentiert. Abbildung 3.3 zeigt zwei Beispielvisualisierungen von Bewegungsmustern.

Der Ansatz verwendet recht allgemeine Verfahren zur Visualisierung der Bewegungsmuster. Die Visualisierung ist sehr abstrakt und es ist schwer für den Arzt einen Bezug zur Physiologie der Bewegungen herzustellen. Es ist daher fraglich, ob die Visualisierungen einen Mehrwert bei der GM Beurteilung bieten. Die daraus extrahierten quantitativen Parameter sind speziell auf die Visualisierungen zugeschnitten und es ist fraglich, ob sie das Bewegungsmuster für eine verlässliche Klassifikation in einer größeren Population allgemein genug beschreiben.

Tabelle 1 fasst die Eigenschaften der verglichenen Ansätze zusammen. Sie führt auf: welche physiologische Grundlage verwendet wurde, welche quantitativen Parameter extrahiert wurden und wie die Ergebnisse visualisiert wurden.

	Physiologische Grundlage	**Quantitative Parameter**	**Visualisierung**
Diese Arbeit	auffällige Bewegungsmuster	Formalisierung auffälliger Bewegungsmuster	Videoaufnahme, Segmentlisten, Bewegungsparameter, Merkmalswerte und -Zeitreihen, Klassifikationsergebnis
Adde et al	Fidgety Movements	Quantität der Bewegung, Standardabweichung des Zentroids	Videoaufnahme, Bewegungsgraph, Bewegungszentroid
Meinecke et al	subjektive Parameter	Kombination aus acht statistischen und mathematischen Parametern	Bewegungsparameter, Klassifikationsergebnis
Berge et al	auffällige Bewegungsmuster	Formalisierung der Visualisierungen von auffälligen Bewegungsmustern	Videoaufnahme, visualisierungen der Bewegungsmuster

Tabelle 1: Gegenüberstellung der verglichenen Ansätze.

Fazit

Der Ansatz dieser Arbeit verwendet objektive Parameter, die für den Arzt nachvollziehbar sind. Durch die Visualisierung von Segmentlisten und Zeitreihen zusammen mit der Videoaufnahme, erlaubt es dem Arzt einen Bezug zwischen den Parametern und den tatsächlichen Bewegungen herzustellen. Der Ansatz ist um weitere Merkmale erweiterbar, die in die Klassifikation einfließen können.

Der Ansatz dieser Arbeit scheint näher an den Ursachen für den visuellen Eindruck zu sein, den Ärzte bei der GM Analyse gewinnen, als allgemeine Verfahren. Der Ansatz ermöglicht es weitere Erkenntnisse leicht zu integrieren.

4. Konfigurationskomponente

Dieses Kapitel stellt die Konzeption der Konfigurationskomponente und ihre Realiserung vor. Am Ende des Kapitels wird die Komponente evaluiert.

4.1. Konzeption

4.1.1. Anforderungen

Die Konfigurationskomponente soll Konfigurationscripte fehlerfreier und für den Forscher leichter zu verstehen und zu bearbeiten machen (siehe Abschnitt 2.2 „Arbeitsablauf“ auf Seite 15). Dazu soll sie eine übersichtlichere Bearbeitung des Scripts ermöglichen und semantische Unterstützung bieten.

Aus der Problembeschreibung (siehe Abschnitt 2.2 „Problematik“ auf Seite 17) wurden folgende Anforderungen an die Komponente abgeleitet.

1. Die Konfigurationskomponente soll als graphischer Editor realisiert werden, der das direkte Editieren von Konfigurationscripten ermöglicht.
2. Sie soll das Konfigurationscript validieren können, indem sie Fehler erkennt und anzeigt.
3. Sie soll Aufgaben automatisieren, indem sie Teile des Scripts automatisch erzeugt.
4. Es soll möglich sein, Scripte die mit einer älteren Version der Komponente erstellt wurden, an neuere Versionen anzupassen.

4.1.2. Technologiewahl

Ein graphischer Editor besteht aus einem zugrunde liegenden Datenmodell, seiner grafischen Repräsentation und Werkzeugen, die das Editieren des Datenmodells erlauben. Aus diesem Grund können bei der Entwicklung eines graphischen Editors Techniken der modellgetriebenen Softwareentwicklung eingesetzt werden.
Modellgetriebene Softwareentwicklung ist nach Stahl [26] ein Oberbegriff für Techniken, die automatisiert aus formalen Modellen lauffähige Software erzeugen. Dabei werden domänenspezifische Sprachen zusammen mit Codegeneratoren eingesetzt. Domänenspezifische Sprachen sind formale Sprachen, die speziell für eine Domäne entworfen werden. Ein graphischer Editor kann dabei als graphische domänenspezifische Sprache und die zu editierenden Dateien als die zu erzeugende Software aufgefasst werden.

Da das Anwendungssystem des Forschungsprojekts auf .NET[8] basiert (ebenso die Datenschnittstelle und die Algorithmenimplementierung), war es naheliegend die Konfigurationskomponente ebenfalls auf .NET Basis zu entwickeln. Dies erleichtert außerdem die Interaktion der Komponente mit den Algorithmen.

[8]Softwareplattform von Microsoft mit einer Laufzeitumgebung und Klassenbibliothek.

Es standen dabei zwei Möglichkeiten für die Entwicklung der Komponente zur Auswahl:

1. Eigene Entwicklung durch den Einsatz der Präsentationskomponenten des .NET Frameworks.
2. Einsatz des Visual Studio Visualization and Modeling SDKs (VSVM SDK).

Da das VSVM SDK bereits einen Großteil der Funktionalität eines graphischen Editors implementiert und außerdem von Microsoft unterstützt und weiterentwickelt wird, fiel die Wahl auf dieses Framework.

VSVM SDK

In der .NET Umgebung existiert ein Framework für die modellgetriebene Softwareentwicklung, das Visual Studio Visualization and Modeling SDK (*früherer Name: DSL Tools*). Das Hauptziel des Frameworks ist es Entwicklern zu ermöglichen, eigene modellbasierte Entwicklungstools innerhalb des Entwicklungswerkzeugs Microsoft Visual Studio[9] zu erstellen und einzusetzen [18] (daher auch sein Name). Es wird von Microsoft unterstützt und weiterentwickelt. Das Framework bietet eine eigene graphische domänenspezifische Sprache zum Definieren des *Metamodells*, also des Modells dass die zu erstellende domänenspezifische Sprache beschreibt. Daraus generiert das Framework mehrere Dinge:

- Implementierung des Metamodells in einer .NET Sprache mit einer streng typisierten Schnittstelle.
- Einen Explorer, der das Modell in einer Baumansicht anzeigt.
- Einen graphischen Editor.
- Serialisierungs- und Deserialisierungscode[10] für die Speicherung des Modells im XML Format.

Jedes dieser Artefakte ist über den integrierten Editor anpassbar und durch benutzerdefinierten Code erweiterbar.
Der generierte graphische Editor ist als Erweiterung für Visual Studio implementiert und benötigt es zum Laufen. Es ist aber auch möglich, diesen in einer kostenlosen Umgebung zu installieren, der Visual Studio Shell. Diese Umgebung beinhaltet die Oberfläche und Grundfunktionalität von Visual Studio, ohne Entwicklungswerkzeuge und Sprachen.
Eine weitere Möglichkeit die, mit VSVM SDK erstellte, domänenspezifische Sprache einzusetzen ist es einen eigenen, WPF basierten Editor zu entwickeln und damit zu integrieren. Einen Ansatz dafür bietet der Artikel [22]. Wir werden hier nicht näher auf diese Möglichkeit eingehen.

Abbildung 4.1 zeigt das Hauptfenster des Metamodell Editors in VSVM SDK und die Grundelemente des Metamodells. Die Ansicht ist in zwei Teile geteilt*: Classes and*

[9] Integrierte Entwicklungsumgebung von Microsoft, die eine Entwicklung von Software auf Basis des .NET Frameworks unterstützt.
[10] Serialisierung bedeutet hier die Abbildung von Objekten auf Text im XML Format.

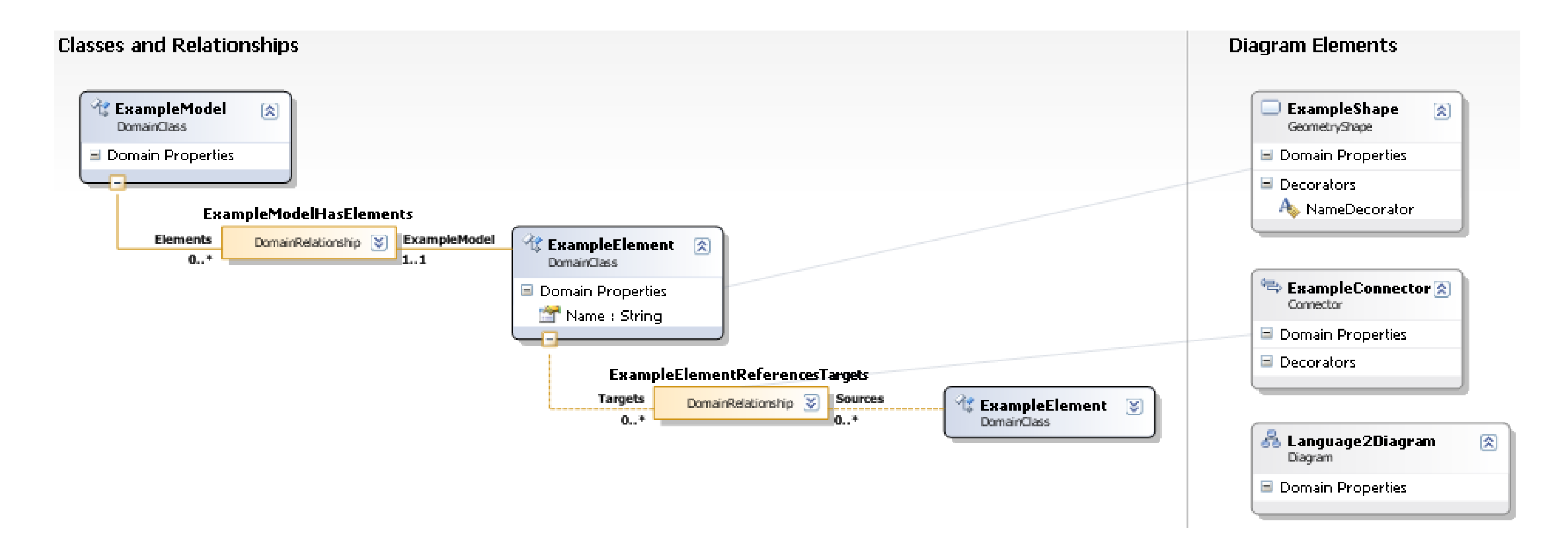


Abbildung 4.1: Bearbeitungsbereich des grafischen Editors für die Erstellung des Metamodells in VSVM SDK

Relationships und *Diagramm Elements*. Die Definition des Metamodells befindet sich im Teil *Classes and Relationships*. Definitionen von grafischen Repräsentationen der Modellelemente im generierten Designer befinden sich im Teil *Diagramm Elements*. Die Zuordnung der Modellelemente zu den grafischen Elementen wird durch Linien zwischen den beiden dargestellt. Die Grundelemente des Metamodells sind folgende:

Domain Class Repräsentieren Elemente der Domäne. Zum Beispiel Personen in einem Familienbaummodell.

Embedding Relationship Repräsentieren Teil-Ganzes Beziehungen in der Domäne. Zum Beispiel Kind-Vater Beziehung in einem Familienbaummodell.

Reference Relationship Repräsentieren Assoziationen in der Domäne. Zum Beispiel Mann-Frau Beziehung in einem Familienbaummodell.

Inheritance Repräsentiert Vererbung im Modell. Dieses Element wird auf die generierte Modellimplementierung abgebildet.

Jedes der Grundelemente kann außerdem Eigenschaften haben. Aus diesen Grundelementen wird das Metamodell des der formalen Sprache zugrundeliegenden Modells aufgebaut. Der Benutzer der Sprache erstellt dann Instanzen des Metamodells.

VSVM SDK ist darauf ausgelegt, grafische domänenspezische Sprachen zu erstellen. Sie sollen von der Implementierung eines Sachverhalts abstrahieren. Die Implementierung kann mithilfe der enthaltenen Technik zur Generierung von textbasierten Dateien anhand von Textvorlagen aus dem Modell generiert werden. Durch Erstellung eigener Vorlagen können beliebige textbasierte Artefakte aus dem Modell erzeugt werden, zum Beispiel: Code, XML Dateien, textuelle Beschreibungen etc.

Weiterführende Erklärungen finden Sie auf den Dokumentationsseiten von Microsoft (siehe [18]) sowie in [3].

4.2. Realisierung

4.2.1. Basisfunktionalität der Komponente

Die Basisfunktionalität der Konfigurationskomponente bilden zusammen: das Metamodell des grafischen Editors (der domänenspezifischen Sprache) und das System zur Extrahierung von Metadaten aus Algorithmen.

Metadaten der Algorithmen

Um Anforderungen 2. und 3. erfüllen zu können, benötigt die Komponente Metadaten[11] der Algorithmen, die im Konfigurationscript eingesetzt werden sollen. Ausgehend von der Definition der Algorithmen in Abschnitt 2.2 „Berechnung von Merkmalen“ auf Seite 14 werden folgende Metadaten benötigt:

[11] hier: Daten, die Informationen über die Algorithmen enthalten.

- Klassenname[12] und Namespace[13] des Algorithmus
- Name und Datentyp der Eingänge
- Name und Datentyp der Ausgänge
- Klassenname und Namespace der Klasse, die die Parameter des Algorithmus enthält
- Name und Datentyp der Algorithmusparameter
- zusätzlich der Name der Merkmale, die der Algorithmus ausgibt

Algorithmen sind als Klassen implementiert. Metadaten von Klassen können in .NET mithilfe von *Reflection* ausgelesen werden. Dies ist eine Technik, die es erlaubt Informationen über Datentypen (Klassen sind ebenfalls Datentypen) zur Laufzeit auszulesen. Allerdings sind einige Informationen implizit in den Algorithmen enthalten und können nicht durch Reflection extrahiert werden. Zum Beispiel erwartet ein Algorithmus bestimmte Eingabedaten. Welche das sind, erfährt man nur aus der Implementierung des Algorithmus, da alle Algorithmen dieselbe generische Schnittstelle besitzen. Soche Informationen müssen explizit deklariert werden. Zu diesem Zweck eignen sich *Attribute*. Attribute in .NET sind eine Möglichkeit, Metadaten deklarativ mit Typen zu assoziieren, diese können dann zur Laufzeit mit Reflection ausgelesen werden.

Um die benötigten Metadaten zur Verfügung zu stellen, wurde eine Klassenbibliothek erstellt, die entsprechende Attribute definiert. Listing 2 zeigt beispielhaft die Definition eines Algorithmus und den Einsatz der Attribute.

Listing 2: Beispiel: Algorithmusdefinition mit Metadaten

```
[Feature("feature1")]
[InputType("in1",typeof(double), false)]
[OutputType("out1", typeof(int), true)]
[InputRelatedType("in2", typeof(ITimeSeries), false,
        "out2_related", typeof(double), RelatedKind.Output,
        "in2_related", typeof(double), RelatedKind.Input)]
public class TrialAlgorithm<TrialAlgorithmParameters>
 : ITrialAlgorithm {
 ...
}

class TrialAlgorithmParameters {
        public double Parameter1 { get; set; }
        public int Parameter2 { get; set; }
}
```

[12]Klasse im Sinne der objektorientierten Programmierung.

[13]Namensraum, in den eine Klasse in der objektorientierten Programmierung eingeordnet wird. Durch den namespace und den Klassennamen ist eine Klasse eindeutig identifiziert.

Im Folgenden wird erläutert wie die Metadaten des Algorithmus aus der Definition im Listing gewonnen werden.

Das Listing stellt eine Klasse dar, die einen Algorithmus definiert, *TrialAlgorithm*.
Der Klassenname und Namespace von TrialAlgorithm können mit Reflection ausgelesen werden. Ebenso der Klassenname und Namespace der Klasse *TrialAlgorithmParameters*. Sie enthält die Parameter des Algorithmus als Eigenschaften[14], deren Namen und Datentypen auch mit Reflection gelesen werden können. Die Klasse wird dem generischen Typparameter[15] von TrialAlgorithm übergeben.
Feature Attribute definieren die Namen der Merkmale, die ein Algorithmus ausgibt.
InputType Attribute definieren die Namen und Datentypen von Eingängen. Der dritte Parameter des Attributs gibt an, ob dieser Eingang mehrere Werte aufnehmen kann.
OutputType Attribute definieren Ausgänge auf dieselbe Weise. *InputRelatedType* Attribute unterstützen ein Feature der Komponente, dass Ausgänge und Eingänge einem bestimmten Eingang zugeordnet werden können und mit ihm zusammen erzeugt und gelöscht werden. *Feature* Attribute erlauben es Namen von Merkmalen anzugeben, die der Algorithmus ausgibt.

Die Komponente lädt die Metadaten eines Algorithmus, sobald er mit einem Script verknüpft wird. Die Verknüpfung wird vom Benutzer der Komponente hergestellt, indem er den Speicherort der Assembly[16] angibt, in der der Algorithmus definiert ist.

Abbildung 4.2 stellt die Objektstruktur, in die Metadaten von Algorithmen geladen werden als Klassendiagramm dar. Diese Struktur erlaubt der Komponente einen schnellen Zugriff auf die Metadaten zur Laufzeit.

Metamodell des Editors

Aus dem Metamodell werden durch das VSVM SDK sowohl der grafische Editor als auch der Serialisierungscode generiert. Um das direkte Editieren von Konfigurationscripten zu ermöglichen, muss das serialisierte Modell als Konfigurationscript eingesetzt werden. Deshalb muss das Metamodell so entworfen werden, dass es beiden Ansprüchen genügt. Die hierarchische Struktur muss der Struktur des Konfigurationscripts entsprechen, während die grafischen Elemente eine sinnvolle Anordnung und Verschachtelung haben müssen.

Abbildung 4.3 stellt vereinfachend die Elemente der Hierarchie des Metamodells dar. Weißer Hintergrund der Elemente bedeutet, dass sie im Konfigurationscript vorkommen. Grauer Hintergrund bedeutet, dass sie nur in der grafischen Darstellung eine Rolle spielen. Die durchgezogenen Linien stellen Verweise zwischen Elementen dar, sie ermöglichen Verbindungen zwischen den Elementen.

[14]Akzessoren in .NET, die den Zugriff auf Datenfelder von Klassen kapseln.
[15]Generische Typparameter sind Konstrukte in .NET, um Klassen zu definieren, die mit unterschiedlichen Datentypen umgehen können. Um eine solche Klasse zu verwenden müssen ihre Typparameter mit konkreten Typen gefüllt werden.
[16]Ausführbare Sammlung der kompilierten Klassen eines Programms.

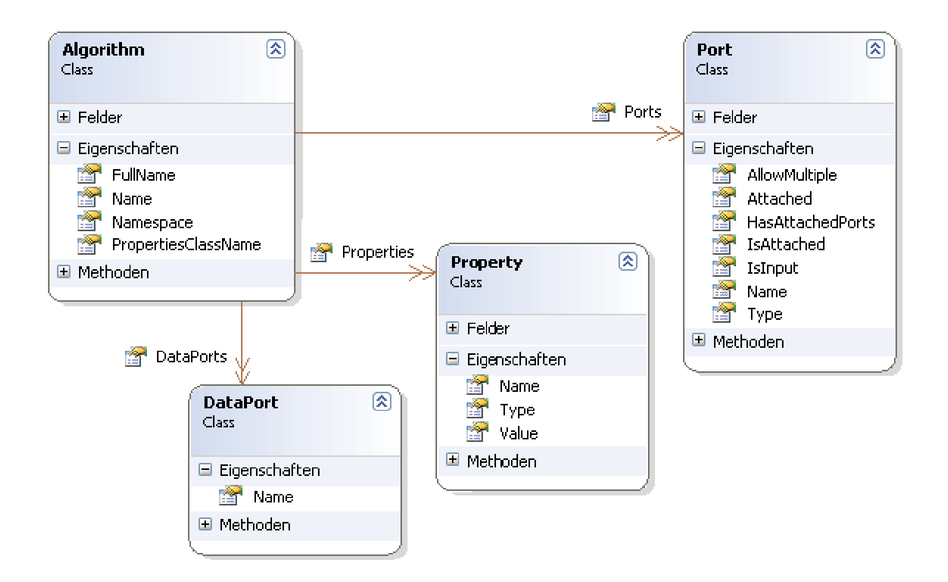


Abbildung 4.2: Klassendiagramm der Objektstruktur zur Verwaltung von Metadaten in der Konfigurationskomponente.

ConfiguratorModel ist das Wurzelelement. Hier werden Verknüpfungen zu den Algorithmen-Assemblys definiert.

VariableGroup, VariableSet und *Variable* sind Konstrukte, die es ermöglichen Variablen als Parameterwerte von Algorithmen festzulegen und die Variablen zu gruppieren.

AlgorithmsSegments gruppiert Algorithm, Segments und Timeline Elemente zur Darstellung in einem eigenen Bereich des grafischen Editors. Es hat ansonsten keine semantische Bedeutung.

Algorithm stellt einen Algorithmus dar. Seine Ein- und Ausgänge werden durch *InputPort* und *OutputPort* dargestellt. *Extremity* stellt Extremitäten dar, die dem Algorithmus zugeordnet werden können (dadurch wird festgelegt, auf welche Aufnahmearten ein Algorithmus angewendet werden kann). *Properties* stellt die Klasse dar, die Algorithmusparameter enthält und gruppiert *Property*, die die Parameter darstellen. *DataPort* stellt Merkmale dar, die ein Algorithmus berechnet. *PortLegend* wird dazu benötigt, eine Legende der Ein- und Ausgänge eines Algorithmus anzulegen, um die Übersichtlichkeit zu erhöhen. Es hat keine semantische Bedeutung.

Segments stellt Einstellungsmöglichkeiten bezüglich Segmentlisten dar.

Timeline stellt eine Zeitreihe dar, die als Eingabe für einen Algorithmus dient.

Database stellt ein grafisches Symbol dar, dass eine Datenbank repräsentiert. Es dient als Verbindungsziel für *DataPort* und hat keine semantische Bedeutung.

Abbildung 4.3: Hierarchische Struktur des Metamodells des Konfigurationscripts. Vereinfachte Darstellung.

Die Elemente Algorithm, InputPort, OutputPort, Property und DataPort leiten ihre Eigenschaften sowohl von den Daten ab, die der Benutzer eingibt als auch von den Metadaten, die ihnen zugeordnet werden. Die Verbindung zwischen den Elementen und ihren Metadaten ist die Basis für Modellvalidierung und Automatisierung. Sie wird beim Erzeugen der Elemente angelegt. Da Metadaten nicht im Konfigurationscript gespeichert werden, muss die Verbindung beim Laden des Konfigurationscripts im Editor wiederhergestellt werden.

Die Verknüpfung wird anhand der Namen und des Kontexts der Elemente hergestellt. Der volle Name (Klassenname qualifiziert mit dem Namen des Namespace) des Algorithmus ist eindeutig. Im Kontext des Algorithmus sind die Namen von InputPort, OutputPort, Property und DataPort ebenfalls eindeutig. Diese Werte werden als Schlüssel genutzt, über die Elemente des Modells mit Metadaten verknüpft werden.

In den nachfolgenden Abschnitten dieses Kapitels werden folgende Begriffe für die Elemente des Modells verwendet:

Algorithmus für das Element Algorithm

Port für die Elemente InputPort und OutputPort

DataPort für das Element DataPorts

Property für das Element Property

4.2.2. Konfigurationscript

Der Editor ermöglicht das direkte Editieren von Konfigurationscripten, indem das editierte Modell als Konfigurationscript gespeichert und aus Konfigurationscripten geladen wird. Dafür muss die Serialisierung des Modells angepasst werden.

Wegen den Anforderungen des VSVM SDK bei der Serialisierung des Modells musste die Struktur des Konfigurationscripts in Teilen geändert werden. Dies wurde bei der Entwicklung inkauf genommen.

Identifizierung der Elemente

Das VSVM SDK erfordert, dass für jedes serialisierte Element, das grafisch dargestellt wird oder auf das andere Elemente verweisen, eine im Kontext des Modells eindeutige Zeichenfolge mitgespeichert wird, die das Element identifiziert, der sogenannte *Moniker*. In der Standardeinstellung wird für jedes Element eine systemweit eindeutige, alphanumerische Zeichenfolge, die sogenannte GUID generiert. Solche Moniker erschweren die Lesbarkeit des Scripts enorm und machen das manuelle Editieren des Scripts schwierig.

Um das Problem zu umgehen, werden eigene Moniker spezifiziert.
Bei VariableGroup, VariableSet und Variable werden ihre *Namen* als Moniker verwendet. Dabei werden die Namen der untergeordneten Elemente jeweils durch die Namen

der übergeordneten qualifiziert, da die Namen nur im Kontext ihres übergeordneten Elements eindeutig sein müssen. Die Eindeutigkeit der Namen wird dabei beim Festlegen programmatisch ermittelt.
Dasselbe Verfahren wird auch bei Timeline angewendet.
Beim Algorithm kann der *Klassenname* als Moniker verwendet werden, da dieser im Kontext des Modells ebenfalls eindeutig ist.
Bei InputPort, OutputPort und DataPort werden *Ids* als Moniker benutzt, die sich aus ihrem Namen und einer Nummer zusammensetzen, die automatisch generiert wird. Das ermöglicht kurze und eindeutige Moniker.
Die übrigen Elemente erfordern entweder keine Identifizierung oder werden durch die automatisch generierten GUIDs identifiziert.

Sortierung der Algorithmen

Die Scriptausführungskomponente führt Algorithmen in der Reihenfolge aus, in der sie im Konfigurationscript definiert sind. Die korrekte Reihenfolge ist sehr wichtig, denn Algorithmen haben durch ihre Verknüpfung Abhängigkeiten voneinander. Wenn beispielsweise Algorithmus A die Ausgabe von Algorithmus B benötigt, muss B vor A ausgeführt werden. Der Benutzer des graphischen Editors hat aber keinen Einfluss darauf, in welcher Reihenfolge die Algorithmen serialisiert werden.

Um dieses Problem zu lösen, wurde ein Verfahren implementiert, das Algorithmen entsprechend ihren Abhängigkeiten korrekt sortiert, die topologische Sortierung. Dafür wurde ein Algorithmus von Lasser verwendet [14]. Im Folgenden wird die Realisierung des Verfahrens dargestellt.

1. Anhand der Verbindungen zwischen den Ports der Algorithmen wird eine Hilfsliste erstellt, die die Algorithmen und ihre logischen Nachfolger enthält.
2. Es wird die Anzahl der logischen Vorgänger jedes Elements in der Hilfsliste berechnet.
3. Alle Elemente ohne Vorgänger werden aus der Hilfsliste entfernt und ans Ende der sortierten Liste hinzugefügt.
4. Schritte 2. und 3. werden solange wiederholt, bis die Hilfsliste keine Elemente mehr enthält.
5. Die sortierte Liste enthält nun die Algorithmen in topologischer Reihenfolge.

Abbildung 4.4 zeigt schematisch die Abhängigkeiten von vier Algorithmen. Hier würden die Algorithmen in folgender Reihenfolge sortiert werden: B, A, D, C (es existieren mehrere gleichwertige Möglichkeiten).

Die Sortierung wird gestartet wenn eine Verbindung zwischen den Ports eines Algorithmus angelegt wird. Sie modifiziert die Reihenfolge der Elemente im internen Speicher. Das bewirkt eine Serialisierung der Elemente in der richtigen Reihenfolge.

Abbildung 4.4: Beispiel: Beziehungen von Algorithmen. Schematische Darstellung.

4.2.3. Automatisierung

Die Basis für Automatisierung bilden Metadaten. Automatisierung in der Komponente wird mithilfe des Regelframeworks von VSVM SDK realisiert.

Regeln beinhalten Code, der ausgeführt wird, wenn ein bestimmten Ereignis auftritt. Regeln werden an Elemente des Modells gebunden. Es existieren verschiedene Typen von Regeln, jeder Typ wird bei einem anderen Ereignis ausgeführt. Zum Beispiel wird eine Regel vom Typ AddRule, die an einen Algorithmus gebunden ist, ausgeführt wenn ein Algorithmus dem Modell hinzugefügt wird.

Gebundene Ein- und Ausgänge

In Abschnitt 4.2.1 „Metadaten der Algorithmen“ auf Seite 28 wurde das InputRelatedType Attribut eingeführt, mit dem eine Bindung von Ports an einen Eingangsport möglich ist. Die Metadaten von Ports, die durch die Metadatenklasse Port repräsentiert werden (siehe Abbildung 4.2) spiegeln diese Bindung in ihren Eigenschaften wieder.

HasAttachedPorts liefert **true**, wenn der entsprechende Eingangsport gebundene Ports hat.

IsAttached liefert **true**, wenn der entsprechende Port selbst gebunden ist.

Attached liefert eine Liste aller gebundener Ports des entsprechenden Eingabeports.

Die Komponente nutzt diese Daten, um automatisch alle gebundene Ports eines Eingabeports zu erzeugen, wenn dieser erzeugt wird, oder zusammen mit ihm zu löschen. Gebundene Ports werden von der Komponente besonders behandelt. Es ist nicht möglich sie manuell zu löschen oder einzeln hinzuzufügen.

Algorithmus Grundgerüst

Die Komponente nutzt die Metadaten des Algorithmus, um bei seiner Erzeugung automatisch alle für ihn definierten Ports, DataPorts und Properties anzulegen, ein Grundgerüst.

Dabei nutzt sie die Eigenschaften der Metadatenklasse Algorithm (siehe Abbildung 4.2): DataPorts, Ports und Properties. Diese liefern ein assoziatives Array[17] (Dictionary in .NET) der jeweiligen Metadatenklassen, die mit ihren Namen identifiziert werden.

4.2.4. Validierung

Das VSVM SDK enthält ein integriertes Framework für die Validierung der Modelle. Es enthält bereits allgemeine Validierungsregeln, die automatisch ausgeführt werden wenn Validierung aktiviert ist. Zum Beispiel wird die Einhaltung der Kardinalitäten[18] von Beziehungen zwischen Elementen des Modells geprüft. Es ist möglich eigene Validierungsregeln für einzelne Elemente des Modells zu implementieren. Listing 3 zeigt beispielhaft die Implementierung einer Validierungsregel. Das *ValidationMethod* Attribut erlaubt die Festlegung, zu welchem Zeitpunkt die Validierungsregel ausgeführt werden soll.

Listing 3: Beispiel einer Validierungsregel

```
[ValidationMethod(ValidationCategories.Open)]
private void ValidatePortConnections(ValidationContext context)
{
    ...
    if(error)
        context.LogError("error text", "category",
            element);
}
```

Typübereinstimmung

Es wurde eine Validierungsregel implementiert, die beim Laden des Modells überprüft, ob die Typen von verbundenen Ports kompatibel sind.

Dazu werden alle Verbindungen zwischen Ports gesucht und ihre Typen auf beiden Seiten verglichen.

[17] In der Programmierung, ein Datenfeld (array) auf dessen Elemente über einen nichtnumerischen Schlüssel zugegriffen werden kann.

[18] Bezeichnet hier die Anzahl der Elemente, mit denen ein Element des Modells in Beziehung stehen muss oder kann.

Zyklenfreiheit

Es wurde eine Validierungsregel implementiert, die beim Laden des Modells die Verbindungen zwischen den Algorithmen über ihre Ports auf Zyklenfreiheit prüft.

Dies wurde durch die, bereits aus Abschnitt 4.2.2 „Sortierung der Algorithmen“ auf Seite 34 bekannte, topologische Sortierung realisiert. Dabei werden die Algorithmen topologisch sortiert, wenn die Verbindungen der Algorithmen einen Zyklus enthalten, kann die Sortierung nicht durchgeführt werden und ein Fehler wird gemeldet.

4.2.5. Fehlervermeidung

Fehlervermeidung unterstützt den Anwender, indem während einer Benutzeraktion geprüft wird, ob sie zulässig ist und die Aktion daraufhin erlaubt oder abgebrochen wird.

Fehlervermeidung bei Verbindungen zwischen Elementen kann im VSVM SDK durch Implementierung von Prüfmethoden der Connection Builder realisiert werden. Als Connection Builder werden Klassen in VSVM SDK genannt, deren Methoden vom grafischen Editor genutzt werden, wenn eine Verbindung zwischen Modellelementen erstellt werden soll. Das Framework generiert automatisch einen Connection Builder für jede Beziehung zwischen Elementen, die im Metamodell definiert wird. Connection Builder können erweitert werden, indem Prüfmethoden implementiert werden. Diese Methoden erlauben es dynamisch zu bestimmen, ob eine Verbindung zwischen den zu verbindenden Elementen erlaubt werden soll.

Typübereinstimmung

Analog zu Abschnitt 4.2.4 „Typübereinstimmung“ auf Seite 36 werden beim Verbindungsversuch zwischen Ports ihre Typen auf Übereinstimmung geprüft.

Die Prüfung erfolgt innerhalb einer Prüfmethode des zugehörigen Connection Builders. Die Verbindung wird nicht erlaubt, wenn die Typen nicht übereinstimmen.

Zyklenfreiheit

Analog zu Abschnitt 4.2.4 „Zyklenfreiheit“ auf Seite 37 wird beim Verbindungsversuch zwischen Ports geprüft, ob diese Verbindung zu einem Zyklus in den Algorithmenverbindungen führen würde.

Die Prüfung erfolgt innerhalb einer Prüfmethode des zugehörigen Connection Builders. Für die Zyklusprüfung wird eine Tiefensuche im Algorithmenbaum durchgeführt.

Die Tiefensuche ist ein Verfahren in der Informatik um Baumstrukturen zu durchlaufen. Dabei wird immer der erste Nachfolgeknoten des Baums expandiert und auf diese Weise nach und nach in die Tiefe gesucht, bis ein Blatt erreicht wird. Danach wird die

Abbildung 4.5: Definition der Schnittstelle für Versionskonverter..

Suche beim nächsten Nachfolger des Vorgängers weitergeführt bis das gesuchte Element gefunden oder alle Knoten des Baums durchlaufen wurden.

Bei diesem Verfahren wird die Tiefensuche bei dem Algorithmus gestartet, der das Ziel der Verbindung ist. Von dort aus durchläuft sie alle Algorithmenverbindungen und sucht nach dem Algorithmus, der die Quelle der Verbindung ist. Wird er gefunden, dann würde die Verbindung zu einem Zyklus führen und wird nicht erlaubt.

4.2.6. Versionskonvertierung

Im Laufe der Weiterentwicklung der Konfigurationskomponente können sich Änderungen am Metamodell ergeben. Diese Änderungen haben auch Auswirkungen auf die Scriptausführungskomponente. Sie muss daran angepasst werden um neue Konfigurationscripte verarbeiten zu können. Konfigurationscripte, die mit einer älteren Version der Komponente erstellt wurden, können dann weder von der Konfigurationskomponente noch von der Scriptausführungskomponente verarbeitet werden.

Um das zu vermeiden ist es erforderlich dass ältere Konfigurationscripte in neuere Versionen konvertiert werden können. Aus Zeitgründen konnte im Rahmen dieser Arbeit kein generischer Versionskonverter implementiert werden. Statt dessen wird eine Schnittstelle angeboten, die genutzt werden kann, um Versionskonverter zu implementieren und in die Komponente zu integrieren. Die so bereitgestellten Konverter werden von der Komponente automatisch aufgerufen, wenn versucht wird ein älteres Konfigurationscript zu laden.

Die Schnittstelle für Versionskonverter ist als Klassenbibliothek implementiert. Abbildung 4.5 zeigt die Methoden der Schnittstelle.

ConvertsActual wird von der Komponente aufgerufen, um zu prüfen, ob der Konverter die vorliegende Version des Scripts verarbeiten kann. Dabei wird seine Versionsnummer übergeben.

ConvertsExpected wird von der Komponente aufgerufen, um zu prüfen, ob der Konverter in die geforderte Version konvertieren kann. Die Versionsnummer wird ebenfalls übergeben.

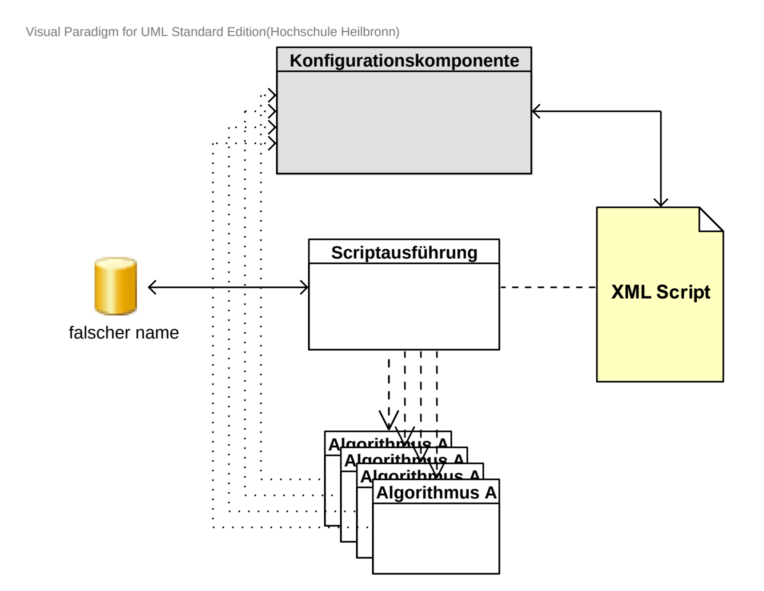


Abbildung 4.6: Anwendung zur Berechnung von Merkmalen erweitert um die Konfigurationskomponente (Schematisch).
Die gepunkteten Pfeile bedeuten dass die Komponente Informationen über Algorithmen ausliest.

Convert ist einfach überladen[19]. Diese Methode wird von der Komponente aufgerufen, um die Konvertierung durchzuführen. Dabei wird entweder nur der Speicherort des Konfigurationscripts oder zusätzlich der Speicherort der Diagrammdatei übergeben.

Versionskonverter werden in die Komponente integriert, indem ihre Assembly in das Verzeichnis für Erweiterungen kopiert wird, dass in der Konfigurationsdatei der Komponente eingestellt ist.

4.3. Evaluation

Die vorgestellte Konfigurationskomponente ergänzt die bestehende Anwendung zur Berechnung von Merkmalen. Abbildung 4.6 zeigt schematisch die Einordnung der Komponente in das bestehende Anwendungssystem.

Vergleich zwischen alter und neuer Vorgehensweise

Im Folgenden wird der Einsatz der Konfigurationskomponente mit dem bisherigen manuellen Verfahren verglichen. Der Vergleich wird anhand eines realen Konfigurationscripts zur Berechnung des Merkmals „Selbstähnlichkeit rechter Arm" durchgeführt. Abbildung 4.7 zeigt die Darstellung des gesamten Konfigurationscripts im Editor. Listing 4 zeigt das Konfigurationscript im XML Format mit der bisherigen Struktur.

[19]Im Sinne der objektorientierten Programmierung. D. h. es existieren mehrere Versionen der Methode mit unterschiedlichen Parameterdefinitionen.

Hier werden nur Auszüge der Funktionalität der Konfigurationskomponente vorgestellt. Für weitere Informationen wird der Leser auf das Benutzerhandbuch im Anhang verwiesen.

Die grafische Darstellung des Konfigurationscripts erlaubt eine gute Übersicht über die Datenflüsse zwischen den Algorithmen. Im Vergleich dazu ist es in der XML Darstellung nicht offensichtlich wie die Algorithmen zusammenhängen. Ein zusätzliches Feature des Editors ist die Darstellung der Datenflüsse zwischen Algorithmen und Datenbank.

Verwendung von Variablen

Listing 5 zeigt die Definition und Verwendung der Variable *var_ Max* im bisherigen Script. Die Variable wird im entsprechenden Abschnitt des Scripts definiert. Dann wird ihre id als Wert beim Parameter eingetragen.
Abbildung 4.8 zeigt die Definition und Verwendung der Variable im Editor. Zuerst werden die VariableGroup, VariableSet und Variable Elemente angelegt. Dann wird der Wert der Variablen im Eigenschaftenfenster eingetragen. In den Eigenschaften der SegmentMaxLength Property kann der Variablenname jetzt unter Value selektiert werden.

Der Editor verhindert durch die Auswahlmöglichkeit von Variablen als Parameterwerte, eine versehentliche Eingabe nichtvorhandener Variablennamen. Der Benutzer kann außerdem dadurch, dass der Property Typ im Eigenschaftenfenster angegeben ist überprüfen, ob der eingegebene Wert oder der Wert der verwendeten Variable korrekt sind. Die XML Darstellung enthält keine Informationen über die Datentypen der Eigenschaften.

Definition von Algorithmen

Listing 6 zeigt die Definition eines Algorithmus im bisherigen Script. Es muss der korrekte Algorithmentyp eingetragen werden. Beim einschließenden Tag der Algorithmusparameter muss der korrekte Name der Klasse angegeben werden.
Abbildung 4.9 zeigt die Definition eines Algorithmus im Editor. Zuerst muss der Algorithmus ausgewählt werden. Der Editor erzeugt danach das Grundgerüst des Algorithmus mit korrekten Definitionen.

Der Editor erleichtert das Anlegen von Algorithmen, indem er eine Auswahl der vorhandenen Algorithmen anzeigt. Das Grundgerüst des Algorithmus mit korrekten Ein- und Ausgängen, Properties und korrekter Definition des Typs wird automatisch erzeugt. In der XML Darstellung muss der Forscher das Grundgerüst manuell eintragen und auf Korrektheit entsprechend der Implementierung des Algorithmus achten.

Verbinden von Algorithmen

Listing 7 zeigt die Verbindung zwischen Ein- und Ausgaben von Algorithmen im bisherigen Konfigurationscript. Die Ausgabe wird benannt. In der Eingabe des referenzierenden

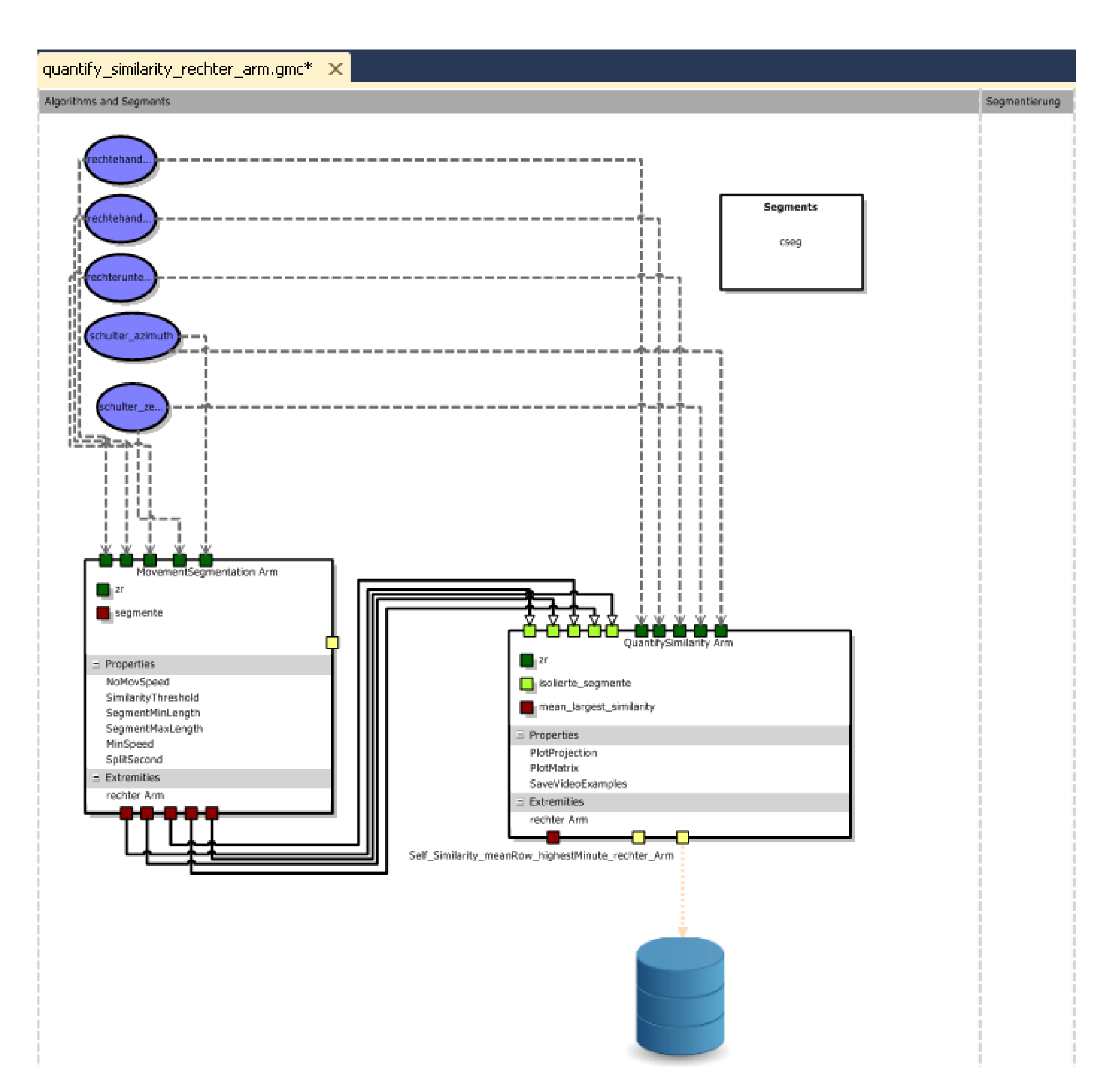


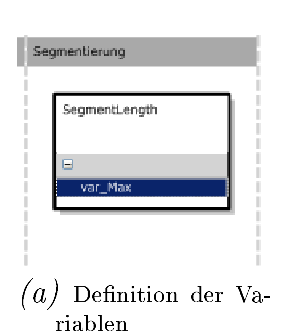
Abbildung 4.7: Darstellung des gesamten Konfigurationscripts im Editor.

Listing 4: Konfigurationscript bisher.

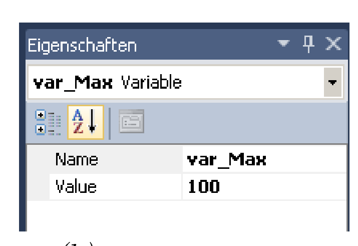
```
<?xml version="1.0" encoding="utf-8"?>
<Skriptliste>
<variablengruppe id="Segmentierung"/>
<skript>
    <segmente>
        <typ>CutSegment</typ>
        <level>0</level>
        <ganzeaufnahme>false</ganzeaufnahme>
        <usefusion>false</usefusion>
        <ausgabe>cseg</ausgabe>
    </segmente>
    <setup id="MovementSegmentation␣Arm">
        <algorithmentyp>
            mocaDataAnalysis.FeatureAnalysis.TrialAlgorithm.MovementSegmentation
        </algorithmentyp>
        <segmente>ref_cseg</segmente>
        <extremitaet>rechter Arm</extremitaet>
        <parameter>
            <MovementSegmentationProperties>
                <NoMovSpeed/>
                <SimilarityThreshold/>
                <SegmentMinLength/>
                <SegmentMaxLength>100</SegmentMaxLength>
                <MinSpeed/>
                <SplitSecond>5</SplitSecond>
            </MovementSegmentationProperties>
        </parameter>
        <zr id="segmente">rechtehandFE</zr>
        <zr id="segmente">rechtehandAA</zr>
        <zr id="segmente">rechterunterarmFE</zr>
        <zr id="segmente">schulter_zenith</zr>
        <zr id="segmente">schulter_azimuth</zr>
        <ausgabe>segmente</ausgabe>
    </setup>
    <setup id="QuantifySimilarity␣Arm">
        <algorithmentyp>
            mocaDataAnalysis.FeatureAnalysis.TrialAlgorithm.QuantifySimilarity
        </algorithmentyp>
        <segmente>ref_cseg</segmente>
        <extremitaet>rechter Arm</extremitaet>
        <parameter>
            <QuantifySimilarityProperties>
                <PlotProjection>false</PlotProjection>
                <PlotMatrix>false</PlotMatrix>
                <SaveVideoExamples>false</SaveVideoExamples>
            </QuantifySimilarityProperties>
        </parameter>
        <zr id="isolierte_segmente">rechtehandFE</zr>
        <zr id="isolierte_segmente">rechtehandAA</zr>
        <zr id="isolierte_segmente">rechterunterarmFE</zr>
        <zr id="isolierte_segmente">schulter_zenith</zr>
        <zr id="isolierte_segmente">schulter_azimuth</zr>
        <zr>ref_segmente</zr>
        <ausgabe>mean_largest_similarity</ausgabe>
    </setup>
</skript>
</Skriptliste>
```

Listing 5: Variablendefinition und Verwendung im Konfigurationscript bisher.

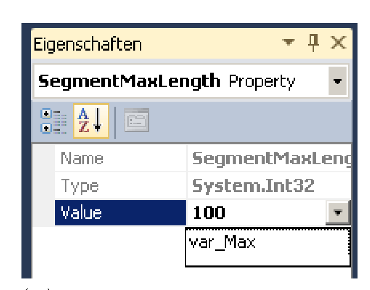
```
<variablengruppe id="Segmentierung">
        <variablenmenge id="SegmentLength">
                <variable id="var_Max">100</variable>
        </variablenmenge>
</variablengruppe>
...
<setup id="MovementSegmentationArm">
        ...
        <parameter>
                ...
                <SegmentMaxLength>var_Max</SegmentMaxLength>
        </parameter>
        ...
</setup>
```



(a) Definition der Variablen

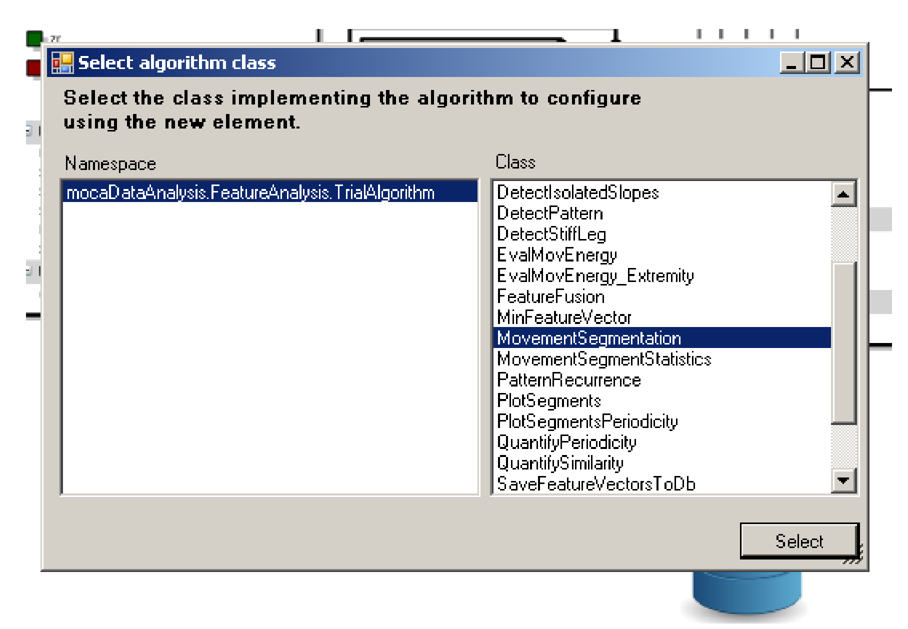


(b) Eintragen des Werts

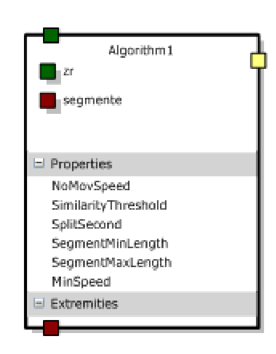


(c) Eintragen der Variable als Parameterwert

Abbildung 4.8: Variablendefinition und Verwendung im Editor.



(a) Auswahl des Algorithmus



(b) Erzeugtes Grundgerüst des neuen Algorithmus.

Abbildung 4.9: Definition eines Algorithmus im Editor.

Listing 6: Definition eines Algorithmus im Konfigurationscript bisher.

```
<setup id="Algorithm1">
        <algorithmentyp>
                mocaDataAnalysis.FeatureAnalysis.TrialAlgorithm.
                    MovementSegmentation
        </algorithmentyp>
        <extremitaet></extremitaet>
        <parameter>
                <MovementSegmentationProperties>
                        <NoMovSpeed></NoMovSpeed>
                        <SimilarityThreshold></SimilarityThreshold>
                        <SplitSecond></SplitSecond>
                        <SegmentMinLength></SegmentMinLength>
                        <SegmentMaxLength></SegmentMaxLength>
                        <MinSpeed></MinSpeed>
                </MovementSegmentationProperties>
        </parameter>
        <zr></zr>
        <ausgabe>segmente</ausgabe>
</setup>
```

Algorithmus wird das Präfix „ref_" vor den Namen der Ausgabe gesetzt.
Abbildungen 4.10 und 4.11 zeigen die Verbindung zwischen den Ports von Algorithmen. Beim Erstellen der Verbindung prüft der Editor den Typ der beiden Ports. Eine Verbindung wird nur zwischen einem Ausgabeport und einem Eingabeport, die kompatible Typen haben zugelassen. Außerdem werden keine doppelten Verbindungen und keine Zyklen erlaubt. In Abbildung 4.11 wird die Verbindung vom Editor nicht erlaubt, da die Typen der Ports nicht kompatibel sind. Abbildung 4.12 zeigt eine Situation, wo die Verbindung der Ports nicht erlaubt wird, weil sie zu einem Zyklus führen würde.

Listing 7: Verbinden von Algorithmen im Konfigurationscript bisher.

```
<setup id="MovementSegmentation_Arm">
         ...
        <ausgabe>segmente</ausgabe>
</setup>
...
<setup id="QuantifySimilarity_Arm">
         ...
        <zr>ref_segmente</zr>
</setup>
```

In XML Darstellung besitzen Ein- und Ausgaben keine Informationen über den Datentyp. Es kann leicht eine Verbindung mit inkompatiblen Typen erzeugt werden. Der Nutzer kann Namen für Ein- und Ausgänge versehentlich doppelt vergeben. Zyklen sind in der

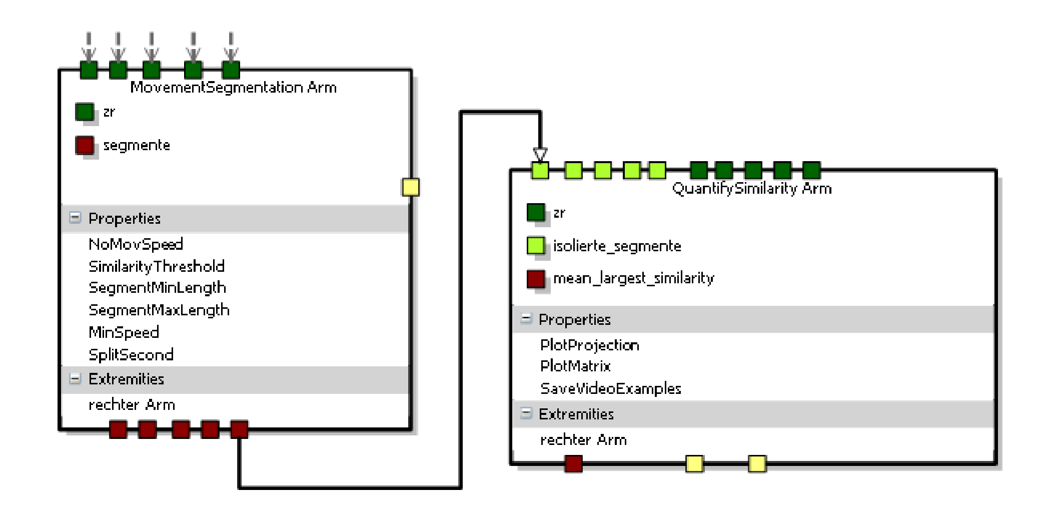


Abbildung 4.10: Verbinden von Algorithmen im Editor, erlaubt.

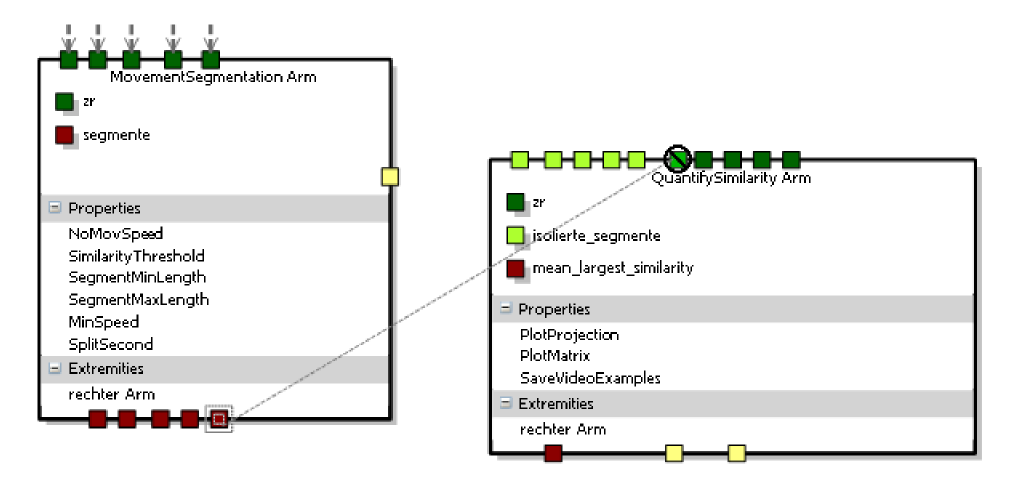


Abbildung 4.11: Verbinden von Algorithmen im Editor, nicht erlaubt.

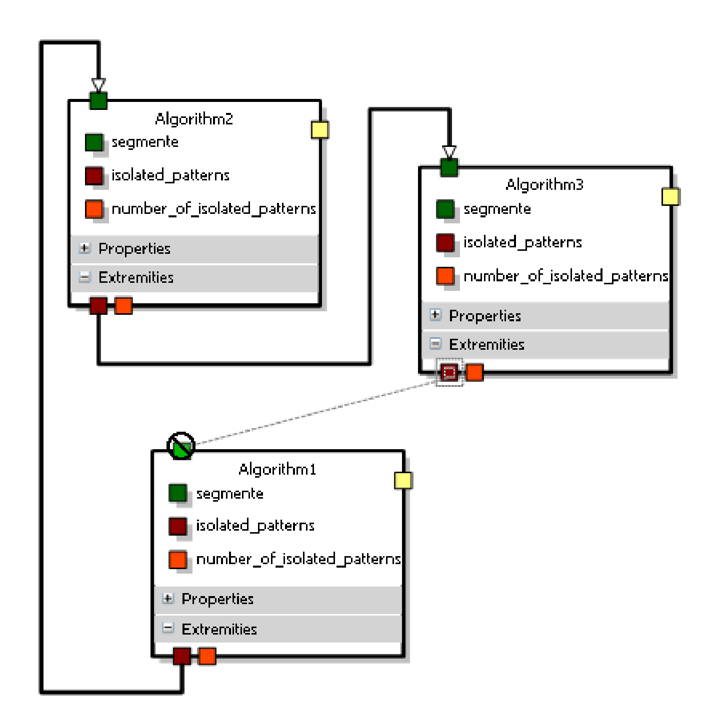


Abbildung 4.12: Verbinden von Algorithmen im Editor, nicht Erlaubt wegen Zyklus.

XML darstellung schwer zu erkennen. Der Editor unterstützt den Benutzer, indem er alle genannten Fehler vermeidet.

Fazit

Die Konfigurationskomponente bietet erhebliche Vorteile gegenüber der manuellen Methode. Sie kann dem Forscher helfen Zeit beim Entwurf von Konfigurationscripten zu sparen und auch über komplexe Scripte die Übersicht zu behalten. Außerdem verhindert die Komponente syntaktische und semantische Fehler und unterstützt den Forscher bei der Fehlersuche.

5. Visualisierungskomponente

Dieses Kapitel stellt die Konzeption der Visualisierungskomponente und ihre Realisierung vor. Am Ende des Kapitels wird die Komponente evaluiert.

5.1. Konzeption

5.1.1. Anforderungen

Die Visualisierungskomponente soll dem Arzt Entscheidungsunterstützung durch Visualisierung objektiver Merkmale bieten. Sie soll auch einen Vergleich zwischen unterschiedlichen Fällen anhand objektiver Merkmale als Vergleichskriterien ermöglichen. Damit soll sie den Schwachstellen der konventionellen GM Beurteilung begegnen (siehe Abschnitt 2.1 „Schwachstellen“ auf Seite 11).

Folgende Anforderungen wurden für die Komponente festgelegt:

1. Die Visualisierungskomponente soll Videoaufnahmen zusammen mit Bewegungsparametern und Merkmalen in übersichtlicher Form darstellen.
2. Es soll möglich sein, Fälle anhand ihrer Merkmalswerte zu klassieren, und dem Arzt das Ergebnis zu präsentieren.
3. Es soll möglich sein, unterschiedliche Fälle anhand der Merkmalswerte zu vergleichen.
4. Es soll möglich sein, Segmentlisten abzuspielen und die Segmente einzeln anzuspringen.
5. Die Visualisierung der Daten soll konfigurierbar sein.
6. Die Komponente soll erweiterbar sein, so dass weitere Merkmale einfach hinzugefügt werden können.

5.1.2. Prototyp

Vor der Implementierung der Komponente wurde ein skizzenhafter Prototyp der Benutzerschnittstelle erstellt und zusammen mit Ärzten des Universitätsklinikums Heidelberg evaluiert.

Die Benutzerschnittstelle teilt den Bildschirm in drei Bereiche ein. Der obere Bereich wird durch die Werkzeugleiste eingenommen. Sie enthält Werkzeuge, mit denen allgemeine Programmfunktionen und layoutbezogene Funktionen (z.B. Ausblenden bestimmter Elemente) ausgeführt werden können. Der Bereich links enthält Detailinformationen. der Bereich rechts enthält den Videoplayer und die Datenvisualisierung.

Die Benutzerschnittstelle wurde so konzipiert, dass sie zwei Ansichten hat. Eine Hauptansicht, die nach dem Start der Anwendung sichtbar ist. Sie enthält Detailinformationen zum Fall und eine Übersicht über die Merkmale. Sowie eine Merkmalsansicht, die detaillierte Informationen zu einem Merkmal anzeigt.

Elemente der Hauptansicht

Abbildung 5.1 zeigt die Hauptansicht der Anwendung im Prototypen. Die Werkzeugleiste enthält:

Das Ordnersymbol Repräsentiert ein Werkzeug zum Laden von Fällen.

Die Pfeile Repräsentieren Werkzeuge zum Navigieren zwischen den Ansichten.

Die Schere und die Grafik Repräsentieren Werkzeuge zum Ein- und Ausblenden von Bereichen.

Der linke Bereich enthält:

Detailinformationen Enthält eine Übersicht über Stammdaten und klinische Daten des aktuellen Kindes.

Übersichten über Merkmale Untereinander angeordnete Übersichten über Merkmale. Jede Übersicht enthält den Namen des Merkmals, seine textuelle Beschreibung und die Einschätzung der Qualität des Merkmals durch das Programm. Dargestellt als rotes (schlechte Qualität) oder grünes (gute Qualität) Viereck.

Der rechte Bereich enthält:

Videoplayer Videoabspielkomponente mit Bedienelementen und Fortschrittsanzeige. Die Bedienelemente enthalten Funktionen zum Abspielen, Pausieren und Stoppen der Wiedergabe. Funktionen, um zum nächsten oder vorherigen Segment zu springen und eine Funktion zum Wiederholen des Videos oder eines Segments. Die Wiedergabegeschwindigkeit kann über den Schieberegler links eingestellt werden. Über den Schieberegler, der die Wiedergabeposition anzeigt, kann an eine andere Stelle im Video gesprungen werden.

Segmentliste Komponente zum Laden und Anzeigen von Segmentlisten. Die Auswahlbox enthält die Namen der Segmentlisten, die zur Auswahl stehen. Über die Checkbox kann die geladene Segmentliste aktiviert oder deaktiviert werden.

Zeitreihen Enthält verschiedene Zeitreihen der Bewegungsparameter untereinander.

Elemente der Merkmalsansicht

Abbildung 5.2 zeigt die Merkmalsansicht der Anwendung im Prototypen. Der linke Bereich enthält jetzt:

Kopfzeile des Merkmals Zeigt den Namen des Merkmals an. Zeigt auch die Einschätzung der momentanen Qualität der Merkmalswerte an der aktuellen Aufnahmeposition.

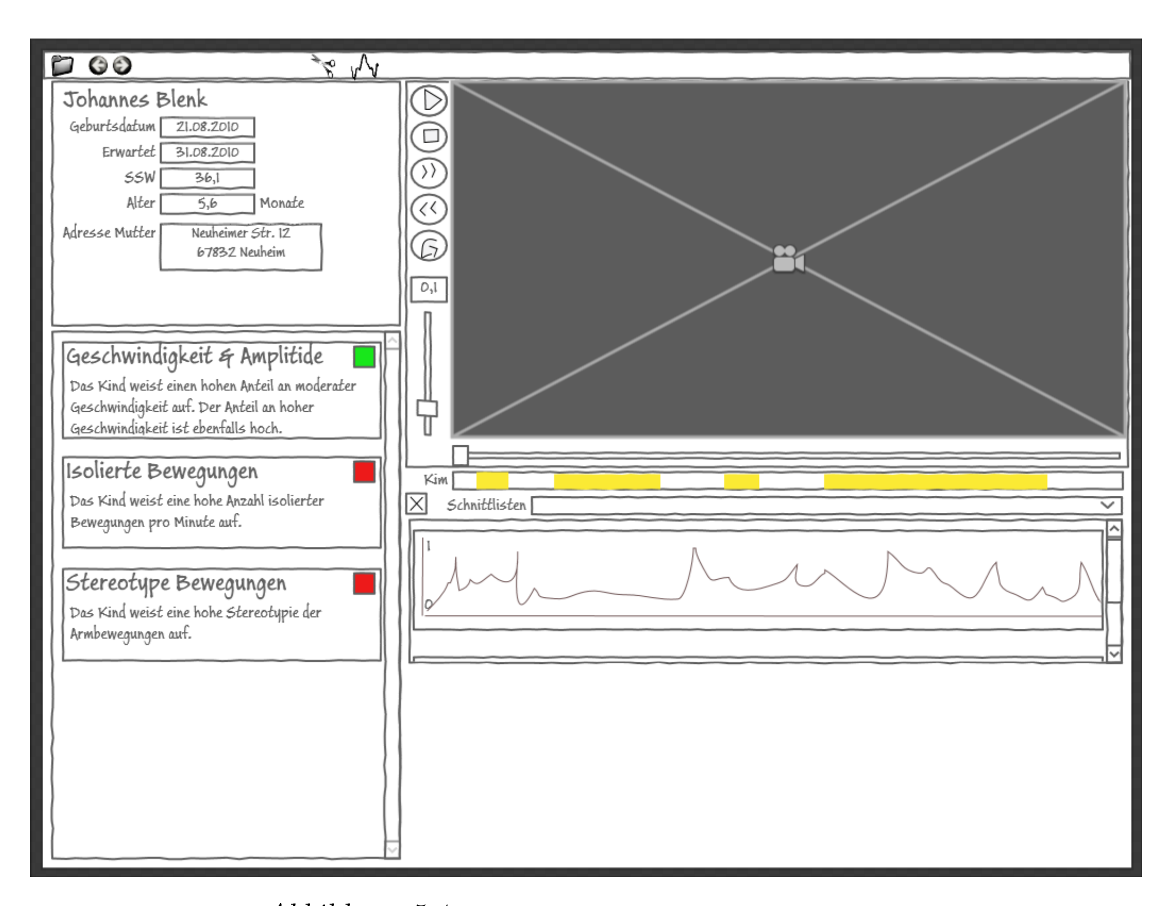


Abbildung 5.1: Prototyp: Hauptansicht der Anwendung.

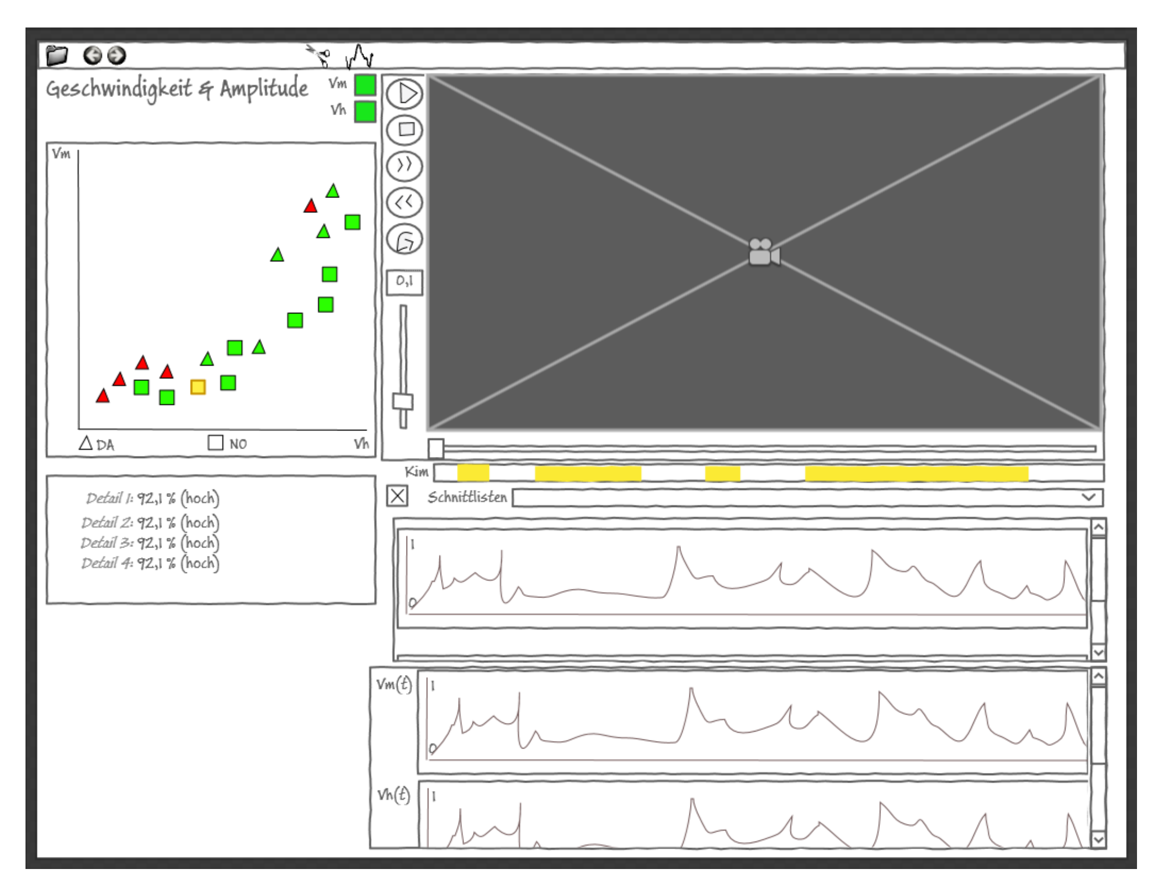


Abbildung 5.2: Prototyp: Merkmalsansicht der Anwendung.

Merkmalsgraph Zeigt die Verteilung der Merkmalswerte bei allen bekannten vergleichbaren Fällen an. Zeigt auch den GM Befund und den Outcome der Fälle an. Das gelbe Rechteck symbolisiert den aktuellen Fall.

Details Zeigt Detailinformationen zum Merkmal und eventuelle Kennzahlen an (Beim Prototypen nicht näher spezifiziert).

Der rechte Bereich enthält zusätzlich zu den Elementen aus der Hauptansicht die Zeitreihen des Merkmals.

Ergebnis der Evaluation

Als Ergebnis der Evaluation wurde der Entwurf der Benutzerschnittstelle verfeinert.

- Die einzelnen Elemente der Benutzeroberfläche sollen einzeln ausblendbar sein. Zu diesem Zweck wurde die Werkzeugleiste um Werkzeuge zum Ausblenden der Elemente: *linker Bereich, Videoplayer, Segmentlisten, Zeitreihen, Zeitreihen des Merkmals* erweitert.

- Der Bereich *Detailinformationen* soll nur folgende Daten anzeigen: Name des Kindes, Geburtsgewicht, Schwangerschaftswoche bei der Geburt, Alter auf der aktuellen Aufnahme.
- Der *Merkmalsgraph* sollte erlauben die dargestellten Fälle direkt zu laden.

Im Laufe der Entwicklung kamen weitere Anforderungen hinzu, die den Entwurf erweiterten.

- Es sollte möglich sein nicht nur eine Einschätzung der Qualität der einzelnen Merkmale vorzunehmen, sondern auch die einer gesamten Aufnahme anhand aller Merkmale. Dazu wurde im Bereich *Detailinformationen* ebenfalls ein Rechteck eingefügt, dass die geschätzte Qualität der Aufnahme darstellt (wie in *Übersicht über Merkmale*).
- *Detailinformationen* sollte zu Orientierungszwecken auch die geladenen Aufnahme und Teilaufname anzeigen. Dazu wurden die entsprechenden Anzeigen unterhalb der Information über das Kind eingefügt.
- Der rechte Bereich in der Merkmalsansicht sollte einen weiteren Bereich enthalten, der Segmentlisten des Merkmals darstellt. Die Segmentlisten sollten wie im Bereich *Segmentliste* aufgebaut sein aber mehrere Segmentlisten auf einmal anzeigen können. Dieser Bereich wurde unterhalb von *Segmentlisten* eingefügt. In der Werkzeugleiste wurde außerdem ein Werkzeug eingefügt, um diesen Bereich ausblenden zu können.

5.1.3. Technologiewahl

Unabhängigkeit zwischen den Komponenten einer Software ist wichtig, weil ihre Funktionen unterteilt und Schnittstellen vereinfacht werden können. Unabhängige Komponenten sind einfacher zu warten, weil Nebeneffekte durch Änderungen von Entwurf und Code begrenzt sind. Die Fehlerfortpflanzung ist verringert und es ist möglich Plug-In Komponenten zu erstellen ([25, Seite 85]). Diese Eigenschaften sind für die Visualisierungskomponente erwünscht. Aus diesem Grund wurden Techniken der modularen Programmierung bei ihrer Entwicklung eingesetzt.

Modulare Programmierung ist nach Schneider [23] eine Programmiertechnik, bei der ein Programm in logisch abgeschlossene Module aufgeteilt wird. Ein Modul ist eine funktionelle Einheit mit bekannten Eigenschaften. Module sind unabhängig voneinander aber können miteinander kommunizieren. Modulare Programmierung basiert auf zwei wichtigen Entwurfsprinzipien der Programmierung: Trennung der Zuständigkeiten (Separation of Concerns) und lose Kopplung (Loose Coupling).

Trennung der Zuständigkeiten Das Prinzip besagt, man soll Teile des Softwaresystems, die für bestimmte Angelegenheiten, Aspekte und Aufgaben verantwortlich sind, identifizieren und als eigene Systembausteine kapseln. So wird das System in verständliche und handhabbare Einzelteile zerlegt (aus [27, Seite 137 f.]).

Lose Kopplung Die Beziehungen der Bausteine eines Softwaresystems zueinander werden als Kopplung bezeichnet. Das Prinzip besagt, die Kopplung möglichst niedrig zu halten. Dadurch wird ein System einfacher veränderbar (aus [27, Seite 131 f.]).

Bei der Entwicklung der Visualisierungskomponente wurde *Prism* eingesetzt, ein Framework dass bei der Entwicklung modularer Anwendungen unterstützt.

Prism

Für .NET 4 und WPF basierte Anwendungen gibt es das Prism Framework [19], dass bei der Entwicklung von modularen Anwendungen unterstützt. Es wird von der Microsoft patterns & practices group unterstützt und weiterentwickelt. Es enthält Leitlinien zu Entwurfsmustern[20], die dabei helfen die Prinzipien Trennung der Zuständigkeiten und lose Kopplung umzusetzen. Außerdem enthält es Referenzimplementierungen der Entwurfsmuster und Klassenbibliotheken, die bei der Umsetzung helfen.

Module

Module in Prism werden durch Klassen identifiziert, die die Schnittstelle[21] *IModule* implementieren und mit dem *ModuleExport* Attribut gekennzeichnet sind. Die Schnittstelle definiert eine Methode *void Initialize()*, sie kann Code enthalten, der bei der Initialisierung des Moduls ausgeführt werden soll. Im *ModuleExport* Attribut können optional Abhängigkeiten von anderen Modulen definiert werden. Prism berücksichtigt diese beim Laden, indem es die Abhängigkeiten zuerst lädt. Es kann auch angegeben werden, ob ein Modul sofort oder erst wenn es gebraucht wird geladen werden soll.

Trennung der Zuständigkeiten bei UI Komponenten

Trennung der Zuständigkeiten bei Komponenten der grafischen Benutzerschnittstelle bedeutet, Trennung des Codes der die grafische Anzeige erzeugt vom Code, der die Anwendungslogik implementiert. Prism unterstützt diese Aufgabe durch das MVVM Entwurfsmuster. Folgende Darstellung ist aus [19] entnommen.

Das MVVM Entwurfsmuster ist eng verwandt mit dem Presentation Model [6] Entwurfsmuster. Es wurde optimiert um die Möglichkeiten von WPF wie data binding, data templates, commands und behaviours zu nutzen (weitere Informationen zu WPF und seinen Möglichkeiten, siehe [15]). Das MVVM (*Model-View-ViewModel*) Muster teilt die UI Komponente in drei Teile (vgl. Abbildung 5.3):

Die View kapselt die grafische Anzeige und Logik, die die grafische Anzeige betrifft. Interagiert mit dem View Model durch data binding, commands und events.

[20] Lösungsschablonen für Entwurfsprobleme in der Softwarearchitektur.

[21] Hier ist eine Schnittstelle in .NET gemeint, die einer abstrakten Klasse im Sinne der objektorientierter Programmierung vergleichbar ist.

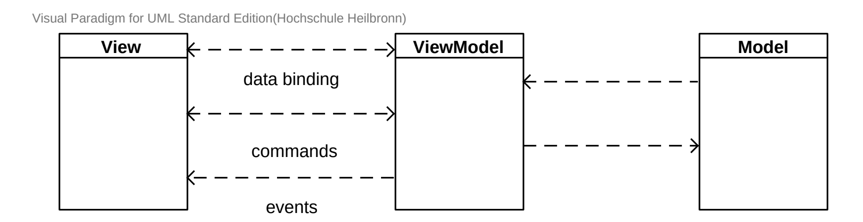


Abbildung 5.3: Elemente im MVVM Entwurfsmuster und ihre Beziehungen zueinander.

Das View Model kapselt Zustandsinformationen und die Logik, die die anzuzeigenden Daten betrifft. Fragt ab und beobachtet das Model. Koordiniert Aktualisierungen des Models. Aggregiert und konvertiert Daten, die in der View angezeigt werden sollen.

Das Model kapselt Anwendungsdaten und -Logik.

Kommunikation und Abhängigkeiten zwischen lose gekoppelten Komponenten

Lose gekoppelte Komponenten (wie Module in einer modularen Anwendung) kennen bestenfalls nur die Schnittstelle anderer Module oder sie kennen sie überhaupt nicht. Damit sie trotzdem kommunizieren können, bietet Prism unter anderem folgende Entwurfsmuster.

Event Aggregator Der Event Aggregator ist eine zentrale Komponente, die Ereignisse mehrerer Komponenten sammelt und an andere Komponenten weiterleitet. Dies ermöglicht es Komponenten, die einander nicht kennen über gemeinsame Ereignisse miteinander zu kommunizieren. Nähere Beschreibung dieses Musters finden Sie in [8].

Gemeinsame Dienste Zentrale Komponenten, die von mehreren Komponenten gemeinsam genutzt werden, die dort Daten ändern und lesen.

Wenn die einzelnen Komponenten zentrale Komponenten wie gemeinsame Dienste und den Event Aggregator nutzen, besitzen sie Abhängigkeiten von diesen. Um diese Abhängigkeitskopplung zu reduzieren, bietet es sich an, einen Dependency Injection Container einzusetzen. Ein DI Container verwaltet die Erstellung und den Lebenszyklus von Komponenten und erfüllt die Abhängigkeiten von Komponenten, ohne dass diese sich darum kümmern müssen. Weitere Informationen über Dependency Injection finden Sie in [7].

Ein Framework dass in Prism als DI Container genutzt wird, ist MEF (*Managed Extensibility Framework*) [17]. Das Framework hilft dabei, PlugIn-fähige Anwendungen

in .NET zu entwickeln. Es kann aber auch als DI Container eingesetzt werden. Das Framework wird von Microsoft unterstützt und weiterentwickelt. Folgende Darstellung ist aus [17] entnommen.

Vereinfachend gesagt, bilden den Kern von MEF Kataloge und der *CompositionContainer*. In einem Katalog kann definiert werden, woher Komponenten geladen werden sollen. Es kann sich um direkt referenzierte Assemblies, um Dateien oder Verzeichnisse handeln. Der *CompositionContainer* lädt Komponenten aus den Katalogen und stellt ihre Instanzen anderen Komponenten zur Verfügung. Komponenten registrieren sich selbst im Container und fordern über Verträge Abhängigkeiten vom Container an. Ein Vertrag kann ein Typ oder ein Name sein.

Listing 8: Beispiel: Nutzung von MEF, um einen gemeinsamen Dienst zur Verfügung zu stellen. *CaseService* registriert sich selbst beim Container mit der abstrakten Schnittstelle als Vertrag. Die Klasse *ViewModel1* fordert eine Instanz von *CaseService* über die abstrakte Schnittstelle an.

```
[Export(typeof(ICaseService))]
public class CaseService : ICaseService {
        ...
}

class ViewModel1 {
        [Import]
        public ICaseService service;
        ...
}
```

Listing 8 demonstriert, wie die Abhängigkeit einer ViewModel von einem gemeinsamen Dienst durch DI aufgelöst wird. Dieses Beispiel demonstriert auch den Einsatz von abstrakten Schnittstellen als Vertrag. Dadurch können Komponenten ihre Abhängigkeiten von Schnittstellen definieren, was es möglich macht, die konkreten Implementierungen der Dienste auszutauschen, ohne andere Komponenten zu beeinflussen.

Zusammensetzen der modularen Anwendung

Um eine Anwendung aus den einzelnen Modulen aufzubauen, definiert Prism einen sogenannten Bootstrapper. Dies ist eine Komponente, die als erstes nach dem Anwendungsstart ausgeführt wird und alle Module lädt und integriert. Der Bootstrapper erlaubt dem Anwendungsentwickler die Kontrolle darüber, wie die modulare Anwendung zusammengesetzt wird.

Um die UI Komponenten der Anwendung, die in unterschiedlichen Modulen existieren können, richtig zusammenzusetzen, bieten Prism das Konzept der Regionen. Eine Region ist ein Platzhalter für eine UI Komponente, der zur Laufzeit gefüllt wird. Regionen können

sowohl im Hauptanwendungsfenster (*Shell* in Prism-Termini), als auch von beliebigen UI Komponenten definiert werden. UI Komponenten können auf zwei Arten mit einer Region verknüpft werden.

- Sie können manuell von der Hauptanwendung instanziiert und in eine Region platziert werden.
- Sie können sich über einen gemeinsamen Dienst in einer Region über ihren Namen registrieren und werden vom Framework zur Laufzeit geladen und in der entsprechenden Region platziert.

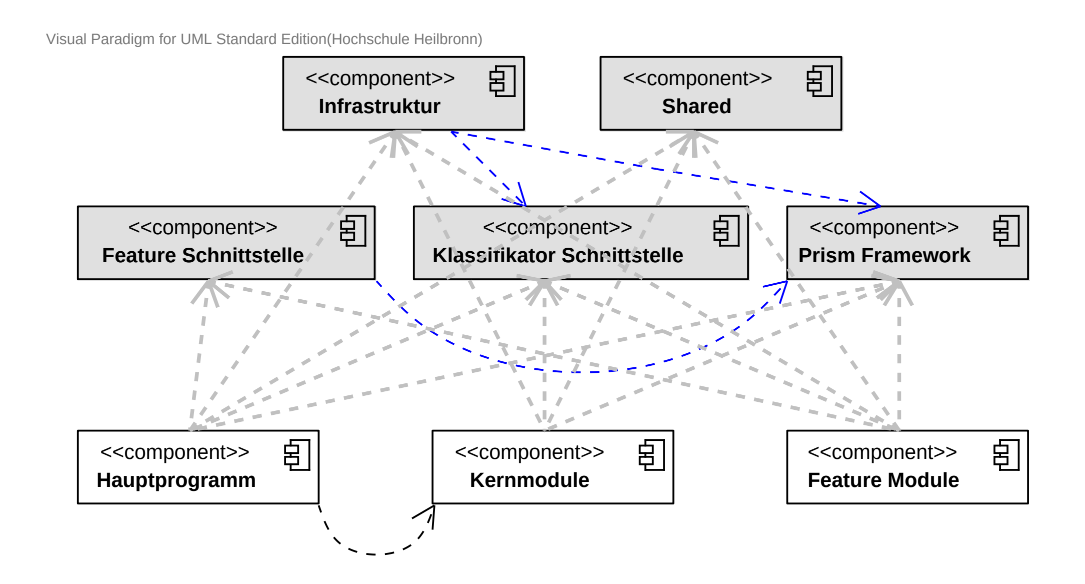


Abbildung 5.4: Komponenten der Visualisierungskomponente und ihre Abhängigkeiten.
Graue Rechtecke: Infrastruktur der Software.
Weiße Rechtecke: Funktionalität der Software.

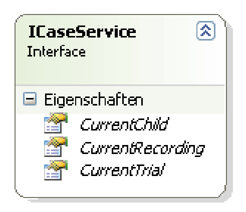
5.2. Realisierung

5.2.1. Architektur

Im Folgenden wird die Architektur der Visualisierungskomponente beschrieben, auf die Realisierung der einzelnen Komponenten wird in späteren Abschnitten eingegangen.

Modularität

Abbildung 5.4 stellt die Module der Visualisierungskomponente und ihre Abhängigkeiten als Komponentendiagramm dar. Weiß hinterlegte Komponenten implementieren die Funktionalität des Systems. Diese Komponenten implementieren sowohl Teile der Anwendungslogik des Systems als auch die entsprechenden Teile der grafischen Benutzerschnittstelle. Grau hinterlegte Komponenten unterstützen die Kommunikation der Komponenten miteinander, es handelt sich um Schnittstellendefinitionen und gemeinsam genutzte Klassen.
Die Abhängigkeiten der grau hinterlegten Komponenten untereinander werden durch dünne blaue Pfeile dargestellt. Die Abhängigkeiten der weiß hinterlegten Komponenten untereinander durch dünne schwarze Pfeile. Die Abhängigkeiten zwischen weiß und grau hinterlegten Komponenten werden durch dicke graue Pfeile dargestellt.



ICaseService erlaubt das Abfragen des aktuell geladenen Falls. *CurrentChild* enthält das aktuelle Kind, *CurrentRecording* die aktuelle Aufnahme (siehe Abschnitt 2.2) und *CurrentTrial* die aktuelle Teilaufnahme.

Abbildung 5.5: Der Dienst ICaseService.

Das Diagramm zeigt dass die weiß hinterlegten Hauptkomponenten des Systems untereinander lose gekoppelt sind. Die einzige Abhängigkeit zwischen den Komponenten *Hauptprogramm* und *Kernmodule* dient der Registrierung der Kernmodule im DI Container (siehe Abschnitt 5.1.3 „Prism" auf Seite 53). Auch die grau hinterlegten Komponenten sind untereinander lose gekoppelt. Die grau hinterlegten Komponenten sind relativ stabil, d.h. sie werden deutlich seltener geändert als weiß hinterlegte. Selbst wenn diese Komponenten erweitert werden, bedingt es nicht die Änderung aller von ihnen abhängigen Komponenten. Erst bei einer Änderung bestehender Definitionen müssen davon anhängige Komponenten auch geändert werden.

Diese Architektur erlaubt es, die Module, die die Funktionalität des Systems implementieren, weitgehend unabhängig voneinander zu entwickeln und zu ändern. Außerdem können Module, die neue Merkmale implementieren leicht ins System integriert werden.

Infrastruktur

Im Folgenden wird näher auf die grau hinterlegten Module aus Abbildung 5.4 eingegangen.

Das Modul *Shared* enthält gemeinsam genutzte Daten. Es definiert die Namen der Regionen in der Shell (siehe Abschnitt 5.1.3 „Prism" auf Seite 53). Außerdem definiert es Stile für die grafischen Elemente der Anwendung. Stile werden in WPF verwendet, um grafischen Elementen ein bestimmtes Aussehen zu verleihen. Sie können global definiert und auf mehrere Elemente angewendet werden.

Das Modul *Prism* bezeichnet das Prism Framework. Es definiert allgemeine Dienste wie den EventAggregator.

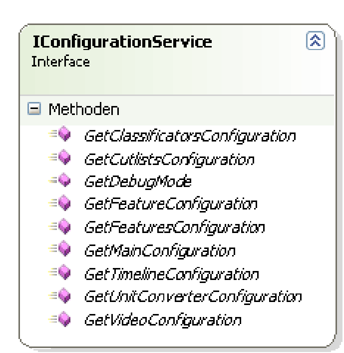
Das Modul *Infrastruktur* enthält Schnittstellen für gemeinsame Dienste, gemeinsam benutzte Model Klassen[22] und gemeinsame Events. Zusammen mit dem Prism Framework bildet es die Hauptkomponente für die Kommunikation zwischen den Modulen des Systems.

Abbildungen 5.5 bis 5.9 zeigen die Schnittstellen der gemeinsamen Dienste (auf die Model Klassen und die gemeinsamen Events wird hier nicht gesondert eingegangen):

[22]Im Sinne des MVVM Entwurfsmusters

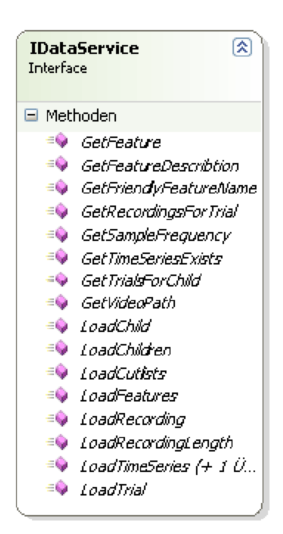
IClassificatorService ermöglicht es Klassifikatoren durch die Angabe ihres Namens zu laden.

Abbildung 5.6: Der Dienst IClassificatorService.



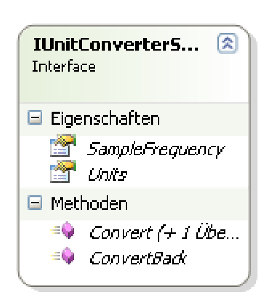
IConfigurationService ermöglicht es verschiedene Konfigurationseinstellungen der Hauptkonfiguration zu laden. Außerdem lädt er den XML Text der Konfiguration für Merkmalsmodule durch Angabe ihres Namens.

Abbildung 5.7: Der Dienst IConfigurationService.



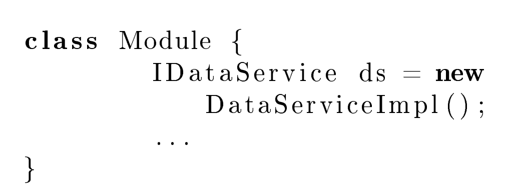
IDataService ermöglicht es Daten, die die Anwendung benötigt aus der Datenbank zu laden.

Abbildung 5.8: Der Dienst IDataService.



IUnitConverterService ermöglicht es Werte zwischen unterschiedlichen Einheiten zu konvertieren.

Abbildung 5.9: Der Dienst IUnitConverterService.



```
class Module {
        IDataService ds = new
            DataServiceImpl();
        ...
}
```

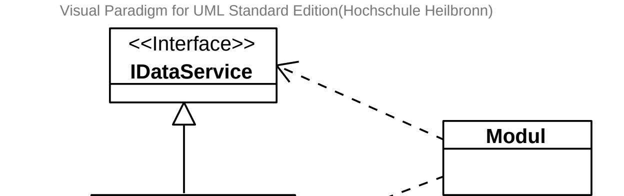


Abbildung 5.10: Direkte Nutzung der Schnittstellenimplementierung durch das Modul.

Schnittstellenimplementierungen

Implementierungen der gemeinsam genutzten Schnittstellen können sich in ganz unterschiedlichen Modulen befinden. Zum Beispiel sind die gemeinsam genutzten Dienste im Modul *Hauptprogramm* implementiert. Deren Schnittstellen aber im Modul *Infrastruktur* definiert. Um die Module mit konkreten Instanzen der Schnittstellen zu versorgen, die sie benötigen, wird ein DI Container eingesetzt.

Die Nutzung eines DI Containers ist notwendig, um die lose Kopplung der Anwendungsmodule zu erhalten.
Abbildung 5.10 demonstriert die direkte Nutzung der Implementierung von *IDataService* durch ein Modul. Das Listing zeigt, wie die Klasse Module eine Instanz der IDataService Implementierung erzeugt. Die Abbildung zeigt die Abhängigkeiten, die sich daraus ergeben. Das Programmmodul das die Klasse Module enthält ist sowohl von dem Modul das IDataService enthält als auch vom Modul das die Implementierung der Schnittstelle enthält abhängig. Damit erbt es auch alle Abhängigkeiten, die DataServiceImpl hat. Somit ist das Prinzip der losen Kopplung verletzt.

Abbildung 5.11 demonstriert den einsatz eines DI Containers. Das Listing zeigt dass die Klasse Module mittels des MEF Attributs *Import* eine Anfrage an den DI Container nach einer Implementierung von *IDataService* stellt. Die Abbildung zeigt die Abhängigkeiten, die sich daraus ergeben. Das Programmmodul das die Klasse Module enthält ist vom Modul das IDataService enthält und vom DI Container abhängig. Nicht aber von der Implementierung der Schnittstelle. Die Verantwortung für das Instanziieren und Verwalten der Schnittstellenimplementierung liegt jetzt beim DI Container.

5.2.2. Hauptprogramm

Das Hauptprogramm bildet den Startpunkt der Anwendung. Es implementiert gemeinsame Dienste, lädt die Module und setzt die Benutzeroberfläche zusammen.

Das Hauptprogramm implementiert einen Bootstrapper, der beim Anwendungsstart den Konfigurationsdienst (IConfigurationService) initialisiert (mehr zur Konfiguration siehe Abschnitt 5.2.5). Danach lädt er die Kernmodule aus den referenzierten Assemblies. Aus

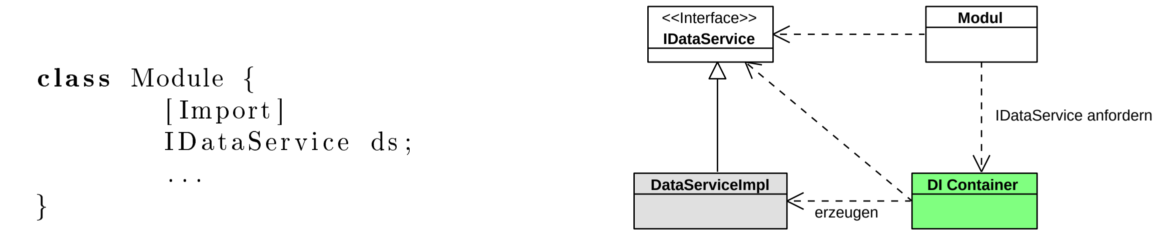


Abbildung 5.11: Einsatz eines DI Containers.

dem Konfigurationsdienst bestimmt er die Verzeichnisse, in denen Klassifikatoren und Merkmalsmodule abgelegt sind und lädt diese ebenfalls.

Das Hauptprogramm implementiert IDataService, indem es die Datenbankschnittstellen der verschiedenen Datenbanken des bestehenden Softwarepakets nutzt (siehe Abschnitt 2.2). Es kapselt die geladenen Daten in gemeinsam genutzten Model Klassen, um die Module der Anwendung von der Datenbankschnittstelle abzuschirmen.

Das Hauptprogramm definiert das Grundgerüst (die Shell) und Grundfunktionalitäten der Benutzeroberfläche. Die Shell definiert ein Layout, dass die Anordnung und Größe der visuellen Elemente am Bildschirm festlegt. Sie definiert auch Werkzeuge für den Benutzer, um das Layout zu ändern, indem: Visuelle Elemente ein- oder ausgeblendet werden (siehe Abschnitt 5.1.2 „Elemente der Hauptansicht“ auf Seite 49), die Größe einzelner Bereiche des Layouts verändert wird.

Die Navigation zwischen der Hauptansicht und der Merkmalsansicht wird vom Hauptprogramm implementiert. Dazu werden die Komponenten des Merkmals (siehe Abschnitt 5.2.4) über ihren Identifizierer geladen und in die entsprechenden Regionen der Shell eingefügt.

Ein Fall wird in der Visualisierungskomponente durch drei Teile repräsentiert: *Child*, *Trial* und *Recording*. Diese Teile sind hierarchisch organisiert. Child repräsentiert ein Kind und ist die oberste Hierarchieebene. Darunter sind entsprechend Abbildung 2.3 in Abschnitt 2.2 *Trial*, das eine Aufnahme repräsentiert und *Recording*, das eine Teilaufnahme repräsentiert.

Eine Grundfunktionalität, die vom Hauptprogramm implementiert wird, ist das Laden von Fällen aus der Datenbank. Diese Funktionalität kann vom Benutzer oder von einem Modul angestoßen werden. Module können anfragen, einen neuen Fall zu laden, indem sie das Event *RequestLoadChildEvent* abfeuern. Wird ein neuer Fall geladen, wird der Falldienst (ICaseService) aktualisiert und entsprechend den geladenen Daten die Ereignisse *ChildChangedEvent*, *TrialChangedEvent* und *RecordingChangedEvent* abgefeuert.

5.2.3. Kernmodule

Die Kernmodule bilden die Kernfunktionalität der Anwendung.

Videoplayer

Das Modul Videoplayer realisiert das Abspielen von Videoaufnahmen. Es ermöglicht außerdem das Abspielen von Segmentlisten und die Anzeige von Informationen zu Segmenten im Videobild.

Das Modul lädt das Video das zur aktuellen Aufnahme gehört aus der Datenbank. Dazu verwendet es die Dienste *ICaseService* und *IDataService*. Wenn sich die aktuelle Aufnahme ändert, lädt das Modul das entsprechende Video. Dazu reagiert es auf den *RecordingChangedEvent*. Die grafische Benutzerschnittstelle des Moduls erlaubt dem Benutzer die Kontrolle über die Wiedergabe. Während der Wiedergabe feuert das Modul in regelmäßigen Abständen das *VideoPositionChangedEvent*, um das System über die aktuelle Wiedergabeposition zu benachrichtigen.

Die Wiedergabe kann durch andere Module des Systems über die *TimelinePositionChangedEvent* und *TimelinePositionChangeCompletedEvent* Ereignisse gesteuert werden. Diese Ereignisse werden beispielsweise ausgelöst, wenn der Benutzer die aktuelle Position in den Zeitreihen ändert (siehe Abschnitt 5.2.3 „Timelines“ auf Seite 64). Diese Ereignisse bewirken, dass im Video an die geänderte Position gesprungen wird.

CutlistLoadedEvent bewirkt die Anzeige von Informationen über Segmente der geladenen Segmentliste im Videobild, wenn die Wiedergabe die Position der Segmentlisten erreicht. Dieses Ereignis wird vom Cutlists Modul (siehe Abschnitt 5.2.3 „Cutlists“ auf Seite 65) ausgelöst wenn eine Segmentliste geladen wird.

Das Abspielen von Segmentlisten wird über die Ereignisse *CutlistSelectedEvent*, *CutlistDeselectedEvent* und *CutSelectedEvent* gesteuert. Wird eine Segmentliste zum Abspielen selektiert, geht der Videoplayer in einen speziellen Abspielmodus über. In diesem Modus stellt die Abspielsteuerung sicher, dass nur Segmente der geladenen Liste abgespielt werden. Das *CutSelectedEvent* bewirkt einen Sprung der Wiedergabeposition zu einem bestimmten selektierten Segment. Das Deaktivieren der Segmentliste schaltet den Abspielmodus wieder auf normale Wiedergabe um.
Abbildung 5.12 stellt dar, wie die Abspielsteuerung vorgeht, wenn während der Wiedergabe einer Segmentliste die Wiedergabeposition manuell geändert wird.

Die Videowiedergabe ist mithilfe der WPF Komponente *MediaElement* implementiert. Diese Komponente basiert intern auf dem Windows Media Player und ermöglicht die Wiedergabe von Videos und die Steuerung der Wiedergabe.
Im Test machte diese Komponente allerdings Probleme bei Videowiedergabe mit verringerter oder erhöhter Wiedergabegeschwindigkeit. Unter anderem wurden folgende Effekte beobachtet:

- Abbruch der Wiedergabe bei längeren Videos nach Erhöhen der Wiedergabegeschwindigkeit.

Abbildung 5.12: Ändern der Wiedergabeposition bei Segmentlisten. Aktivitätsdiagramm.

- Ende der Wiedergabe wird nicht korrekt gemeldet wenn während der Wiedergabe die Geschwindigkeit verändert worden ist. Das Video musste danach erneut geladen werden.
- Einstellung niedriger Wiedergabegeschwindigkeit wurde bei längeren Videos ignoriert.

Da WPF eine relativ neue Technologie ist, besteht die Möglichkeit, dass die genannten Fehler in den nächsten Versionen korrigiert werden. Eine Alternative wäre es, das Abspielen von Videos mithilfe von DirectShow[23] selbst zu implementieren. Aus Zeitgründen wurde dies in der vorliegenden Arbeit unterlassen.
Da das Modul nach dem MVVM Entwurfsmuster implementiert ist, ist es möglich die Videowiedergabe unabhängig von der Modullogik zu implementieren.

Timelines

Das Modul Timelines ermöglicht die Darstellung von Bewegungsparametern als Zeitreihen. Es verfügt über eine Anzeige der aktuellen Wiedergabeposition, eine Möglichkeit die Wiedergabeposition zu verändern und über eine Zoom-Funktion.

Das Modul nutzt die Dienste *IConfigurationService* um sich zu konfigurieren, sowie *IDataService* und *ICaseService* um Zeitreihen für die aktuelle Aufnahme zu laden. Es reagiert auf den *RecordingChangedEvent*, um neue Zeitreihen zu laden, wenn die aktuelle Aufnahme sich ändert. Das Modul kann eine beliebige Anzahl von Zeitreihen darstellen. Welche Zeitreihen das sind, wird in der Konfiguration angegeben (siehe Abschnitt 5.2.5). Der anzuzeigende Bereich der Zeitreihe (auf der X-Achse), sowie die zu verwendenden Einheiten (für die X-Achse, die Y-Achse ist üblicherweise einheitenlos) werden ebenfalls in der Konfiguration festgelegt.

Die grafische Darstellung der Zeitreihen wurde mithilfe der Drittanbieterkomponente *Visiblox*[24] realisiert. Die kommerzielle Komponente implementiert Graphen für Datenvisualisierung in WPF. Sie wurde gewählt, weil sie eine hohe Performance bei der Darstellung bietet und einfach zu benutzen und zu erweitern ist. In dieser Arbeit kommt die kostenlose Version der Komponente zum Einsatz, die ein Wasserzeichen in den Graphen einbettet. Für die Anzeige der aktuellen Wiedergabeposition wurde eine Erweiterung für Visiblox implementiert, die eine manipulierbare Linie in der Darstellung der Zeitreihen einfügt. Die Position der Linie wird vom Modul auf die aktuelle Wiedergabeposition aktualisiert, wenn ein *VideoPositionChangedEvent* eintritt. Wenn der Benutzer die Position der Linie manuell manipuliert, feuert das Modul das *TimelinePositionChangedEvent*, während der Benutzer die Linie manipuliert und das *TimelinePositionChangeCompletedEvent* wenn die Manipulation abgeschlossen ist.

Die Anzeige der aktuellen Wiedergabeposition in den Zeitreihen, hat im Test bei großen anzuzeigenden Bereichen zu Performanceproblemen geführt. Aus diesem Grund wurde

[23]Grafikschnittstelle unter Windows, die die Implementierung von Medienwiedergabe ermöglicht.
[24]`http://www.visiblox.com`

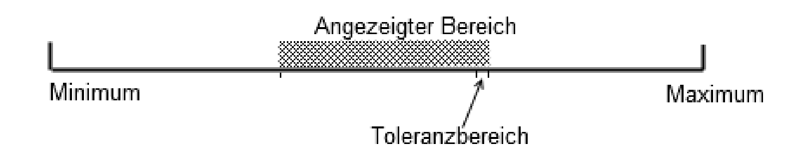


Abbildung 5.13: Anzeige eines begrenzten Bereichs einer Zeitreihe.

eine Funktion implementiert, über die der Benutzer die Anzeige der aktuellen Position ein- und ausschalten kann. Um das Problem zu umgehen, sollte der anzuzeigende Bereich in den Zeitreihen nicht zu groß gewählt werden. Ein Richtwert, der heuristisch bestimmt wurde, ist weniger als 5000 Datenpunkte pro Zeitreihe (bei 50 Hz Abtastfrequenz ergibt das einen 100 Sekunden breiten Bereich). Besonders bei vielen gleichzeitig darzustellenden Zeitreihen sollte der Bereich angepasst werden.

Das Modul verfügt über eine Steuerungslogik, die den angezeigten Bereich der Zeitreihe aktualisiert und Daten für diesen Bereich aus der Datenbank nachlädt, wenn die aktuelle Wiedergabeposition den angezeigten Bereich überschreitet.
Abbildung 5.13 veranschaulicht einen sichtbaren Bereich im Kontext der gesamten Zeitreihe. Wenn die aktuelle Position den angezeigten Bereich in negativer Richtung überschreitet, wird der Bereich geladen, dessen Breite der eingestellten Bereichsbreite entspricht und dessen Maximum bei der neuen Position (oder darüber, wenn der Bereich sonst zu klein würde) liegt. Wenn die Position eine Toleranzgrenze in positiver Richtung überschreitet, wird der Bereich geladen, dessen Minimum bei der Toleranzgrenze liegt.

Cutlists

Das Modul Cutlists stellt Segmentlisten grafisch dar. Es ermöglicht das Laden und Aktivieren / Deaktivieren von Segmentlisten. Außerdem ermöglicht es dem Benutzer, einzelne Segmente auszuwählen.

Das Modul nutzt die Dienste *IConfigurationService* um sich zu konfigurieren, sowie *IDataService* und *ICaseService* um Segmentlisten für die aktuelle Aufnahme zu laden. Es reagiert auf den *RecordingChangedEvent*, um neue Segmentlisten zu laden, wenn die aktuelle Aufnahme sich ändert. Das Modul stellt vorkonfigurierte Segmentlisten zur Auswahl wenn sie für die aktuelle Aufnahme verfügbar sind (siehe Abschnitt 5.2.5).

Für die grafische Darstellung der Segmente wurde *Visiblox* zusammen mit einer eigenen Erweiterung verwendet. Dabei werden Segmente auf einer Zeitskala als Rechtecke repräsentiert, deren Länge der Länge der Segmente in Zeiteinheiten entspricht. Die Segmente sind dabei auswählbar.

Das Modul feuert das *CutlistLoadedEvent*, sobald eine Segmentliste geladen wurde. Es feuert das *CutlistSelectedEvent*, wenn eine Segmentliste aktiviert wurde und das *CutlistDeselectedEvent*, wenn sie deaktiviert wurde. Wird ein Segment ausgewählt, feuert das Modul ein *CutSelectedEvent*.

Das Modul reagiert zusätzlich auf das *CutlistSelectedEvent* indem es die aktive Segmentliste deaktiviert wenn das Ereignis nicht durch ihre Aktivierung ausgelöst wurde. Dadurch wird verhindert, dass in Verbindung mit einem Merkmalsmodul mehrere Segmentlisten gleichzeitig aktiv sind (siehe Abschnitt 5.2.4).

Casedetails

Das Modul Casedetails gibt Stammdaten des aktuellen Kindes aus. Es zeigt auch die Namen der aktuellen Aufnahme und Teilaufnahme. Es aktiviert außerdem den globalen Klassifikator (siehe Abschnitt 5.2.6) für die Bewegungsqualität des Kindes und visualisiert das Ergebnis.

Das Modul nutzt die Dienste *IConfigurationService* um sich zu konfigurieren, sowie *IDataService* und *ICaseService*, um Stammdaten des aktuellen Kindes und Namen der aktuellen (Teil-)Aufnahme zu laden. Es reagiert auf die Ereignisse *ChildChangedEvent*, *TrialChangedEvent* und *RecordingChangedEvent* indem es Daten aktualisiert und die Klassifikation erneut anstößt.

Der Name und die Eingabemerkmale des globalen Klassifikators sind in der Konfiguration definiert (siehe Abschnitt 5.2.5). Das Modul nutzt den Namen, um einen Klassifikator über den *IClassificatorService* Dienst zu laden. Wenn eine neue Aufnahme geladen wird, lädt das Modul die Eigabemerkmale aus der Datenbank und führt damit eine Klassifikation durch.

5.2.4. Merkmalsmodule

Ein Merkmalsmodul hat die Aufgabe ein bestimmtes Merkmal zu visualisieren. Dazu hat das Modul sechs definierte Ansichten zur Verfügung. Diese Ansichten werden an definierten Stellen in Regionen des Hauptprogramms eingefügt. Im Folgenden werden die Ansichten aufgeführt.

FeatureOverview Diese Ansicht bietet eine Übersicht über das Merkmal. Sie wird in Abschnitt 5.1.2 „Elemente der Hauptansicht" auf Seite 49 als *Übersicht über Merkmale* beschrieben. Abbildung 5.14 zeigt schematisch die Hauptansicht der Anwendung und die Position mehrerer FeatureOverview Ansichten darin.

FeatureHeader Diese Ansicht bildet die Kopfzeile der Merkmalsansicht der Anwendung. Sie wird in Abschnitt 5.1.2 „Elemente der Merkmalsansicht" auf Seite 49 als *Kopfzeile des Merkmals* beschrieben.

FeatureGraph Enthält den Merkmalsgraphen. Diese Ansicht wird in Abschnitt 5.1.2 „Elemente der Merkmalsansicht" auf Seite 49 als *Merkmalsgraph* beschrieben. Der Merkmalsgraph kann auch mehrdimensional sein, wenn das Merkmal mehrere Werte enthält. Der Merkmalsgraph ermöglicht das Laden von Fällen durch Selektion im Graphen.

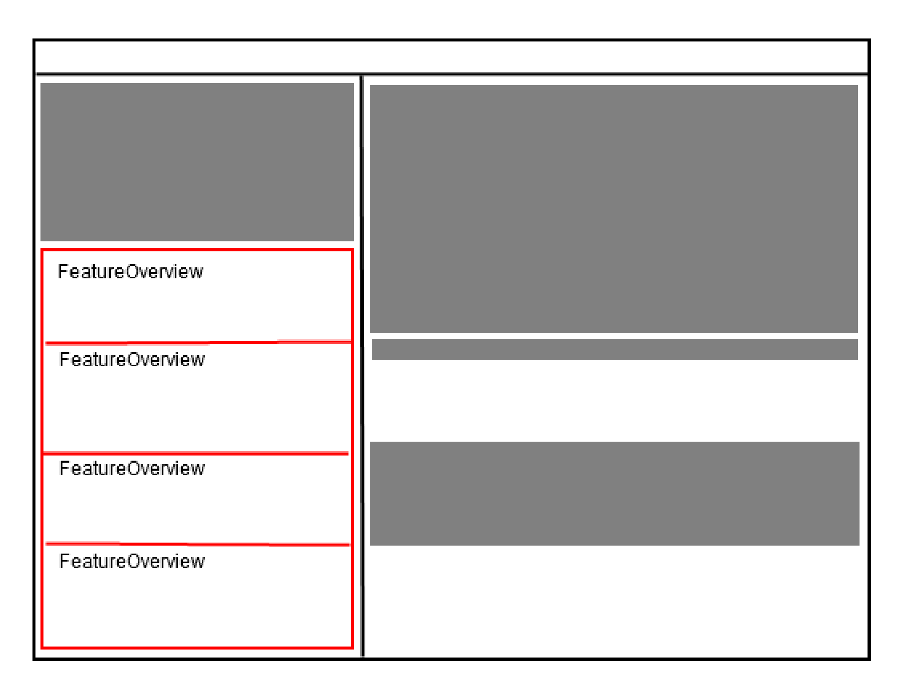


Abbildung 5.14: Platzierung der Merkmalsansichten in der Hauptansicht der Anwendung. Schematisch.

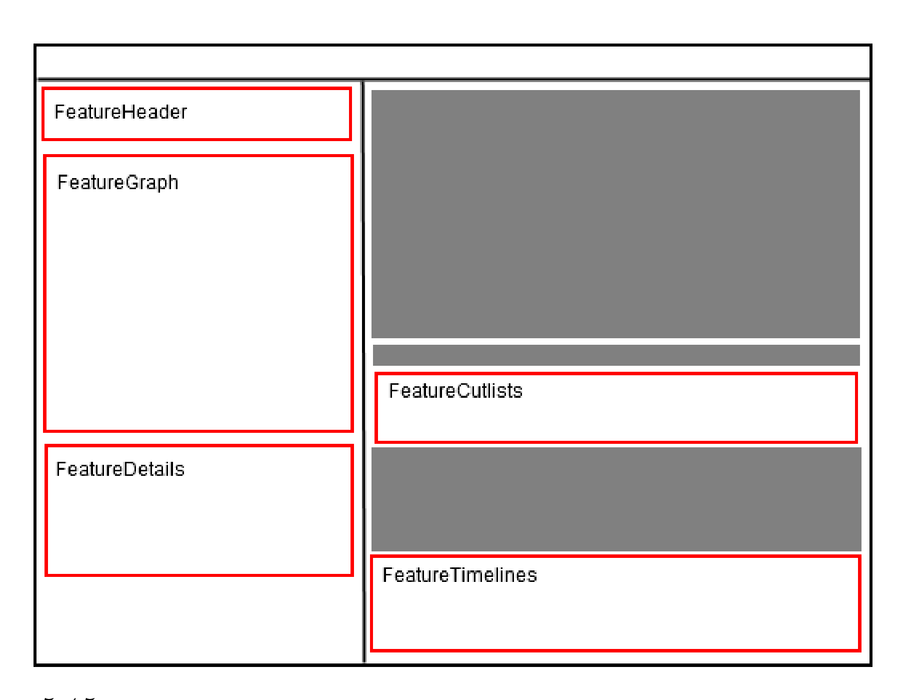


Abbildung 5.15: Platzierung der Merkmalsansichten in der Merkmalsansicht der Anwendung. Schematisch.

FeatureDetails Enthält Detailinformationen über das Merkmal. Diese Ansicht wird in Abschnitt 5.1.2 „Elemente der Merkmalsansicht“ auf Seite 49 als *Details* beschrieben.

FeatureCutlists Enthält Segmentlisten für das Merkmal. Sie weisen dieselben Eigenschaften wie im Modul Cutlists auf (siehe Abschnitt 5.2.3 „Cutlists“ auf Seite 65). Ein Unterschied ist, dass hier mehrere Segmentlisten gleichzeitig angezeigt werden können.

FeatureTimelines Enthält Zeitreihen für das Merkmal. Sie weisen dieselben Eigenschaften wie im Modul Timelines auf (siehe Abschnitt 5.2.3 „Timelines“ auf Seite 64).

Abbildung 5.15 zeigt schematisch die Merkmalsansicht der Anwendung und die Lage der Ansichten des Merkmalsmoduls darin.

Merkmalsmodule werden durch eine Zeichenfolge identifiziert, die unter allen Merkmalsmodulen eindeutig ist. Sie wird in der Modulimplementierung festgelegt und in der Konfiguration und der Anwendung verwendet, um das Modul zu identifizieren. Sie wird in dieser Arbeit als *Identifizierer* eines Merkmals bezeichnet.

Schnittstelle

Die Schnittstelle für Merkmalsmodule ist als Klassenbibliothek implementiert. Sie erleichtert die Registrierung des Merkmalsmoduls im Hauptprogramm. Sie enthält außerdem Attribute und eine Schnittstelle[25], über die die Ansichten eines Merkmalsmoduls identifiziert werden können.

Die abstrakte[26] Klasse *ModuleBase* bildet die Grundlage des Merkmalsmoduls. Diese Klasse implementiert die Schnittstelle *IModule* von Prism und übernimmt die Registrierung der FeatureOverview Ansicht des Merkmals in der entsprechenden Region des Hauptprogramms.

Die Attribute: *FeatureOverview*, *FeatureHeader*, *FeatureGraph*, *FeatureDetails*, *FeatureCutlists* und *FeatureTimelines* identifizieren die Arten der Merkmalsansichten. Die Schnittstelle *IFeatureComponent*, die von jeder Ansicht implementiert werden muss, definiert eine Eigenschaft, die einen eindeutigen Identifizierer des Merkmals zurückgibt.

Referenzimplementierung

Um die Entwicklung von Merkmalsmodulen für neue Merkmale zu vereinfachen, wurden zwei Referenzimplementierungen der Schnittstelle für Merkmalsmodule erstellt. Eine davon enthält die Grundimplementierung der Schnittstelle, die nötig ist, damit das Modul als Merkmal erkannt wird. Die andere enthält die komplette Implementierung auf Basis

[25]Hier ist eine Schnittstelle in .NET gemeint
[26]im Sinne objektorientierter Programmierung

eines bereits vorhandenen Merkmals. Diese Implementierung kann als Ausgangspunkt für neue Merkmale verwendet und entsprechend angepasst werden.

Die Referenzimplementierungen enthalten eine konkrete Klasse, die von ModuleBase abgeleitet[27] ist. Diese Klasse ist mit dem *ModuleExport* Attribut aus Prism versehen, da es nicht auf der Basisklasse deklariert werden kann.

Jede Ansicht des Merkmals ist als UserControl[28] in WPF implementiert und mit einem der oben genannten Attribute gekennzeichnet. Sie ist außerdem mit dem *Export* Attribut von MEF gekennzeichnet und implementiert die Schnittstelle *IFeatureComponent*.

Die Ansichten der kompletten Referenzimplementierung sind analog zu den Kernmodulen implementiert. Eine Ausnahme bildet FeatureGraph. Dieser ist mithilfe von Visiblox und einer eigenen Erweiterung implementiert. Die Erweiterung ermöglicht die Anzeige der Datenpunkte als unterschiedliche Formen mit unterschiedlichen Farben, je nachdem welche Kategorie für die Datenfolge eingestellt wird.
Die Daten, die benötigt werden sind folgendermaßen definiert: alle Fälle mit derselben GM Phase wie der aktuelle Fall, für die das aktuelle Merkmal berechnet wurde, als Tupel aus Fall und Merkmalswert gruppiert nach ihrem GM Befund und ihrem Outcome.
Die Komponente greift direkt über die Datenbankschnittstelle auf die Datenbank zu, um die benötigten Daten zu laden. Da dies eine sehr spezielle Aufgabe ist und keines der Kernmodule der Anwendung solche Daten benötigt, ist diese Funktionalität nicht im Datendienst der Anwendung implementiert.

Integration

Um ein Merkmalsmodul in die Anwendung zu integrieren sind drei Schritte notwendig.

1. Implementieren der Schnittstelle für Merkmalsmodule. Dazu kann die Referenzimplementierung als Ausgangspunkt benutzt oder komplett übernommen werden.
2. Kompilieren des Mermalsmoduls als Assembly und kopieren dieser Assembly ins Verzeichnis für Merkmalsmodule.
3. Eintragen des neuen Merkmals in der Konfigurationsdatei der Anwendung evtl. zusammen mit Konfigurationseinträgen für das Merkmalsmodul (siehe Abschnitt 5.2.5).

5.2.5. Konfiguration

Die Konfiguration der Komponente erfolgt durch eine einzige Konfigurationsdatei. Eine Konfigurationsdatei ermöglicht es die Anwendung vorkonfiguriert auszuliefern und Konfigurationen schnell auf verschiedene Installationen zu verteilen.

[27] im Sinne der Vererbung in der objektorientierten Programmierung
[28] Benutzerdefinierte Zusammensetzung aus verschiedenen grafischen Komponenten in WPF mit eigener Schnittstelle.

Die Konfigurationsdatei liegt im XML Format vor. Das Format der Datei wird durch ein XML Schema[29] bestimmt. Das Schema definiert den Aufbau der Konfiguration für die Hauptanwendung und Bereiche für die Konfiguration der Merkmale, die beliebigen XML Text enthalten dürfen. Auf diese Weise können Merkmalsmodule ihre eigenen Konfigurationsformate definieren, die in die Gesamtkonfiguration eingebaut werden.

Die Konfigurationsdatei wird beim Anwendungsstart von der Anwendung ausgelesen. Dabei werden Bereiche für die Konfiguration der Merkmale direkt als XML Text übernommen. Der Inhalt der Konfigurationsdatei wird in einer abstrahierten Form über einen Dienst (*IConfigurationService*) den Modulen zugänglich gemacht. Merkmalsmodule können dabei den Teil, der für sie bestimmt ist, über ihren Identifizierer abrufen und selbst auslesen.

Im Folgenden wird der Inhalt und die Bedeutung der Konfigurationsdatei erläutert.

Listing 9: Konfiguration. Visualization.

```
<visualization
        xmlns="http://www.uni-heidelberg.de/GMVisualization"
        debug="false" defaultRecording="standardteilaufnahme">
```

Listing 9 zeigt das Wurzelelement der Konfigurationsdatei: *visualization*. Es enthält logisch sortierte Konfigurationsbereiche und globale Einstellungen.
Das Attribut *debug* schaltet den Debug-Modus ein oder aus. In diesem Modus greift die Anwendung nicht auf die Datenbank zu, sondern arbeitet mit Beispieldaten.
Dass Attribut *defaultRecording* ermöglicht die Angabe einer Teilaufnahme die standardmäßig geladen wird, so dass der Benutzer nur das Kind und die Aufnahme wählen muss, wenn er einen Fall lädt.

Listing 10: Konfiguration. Features.

```
<features
        directory="C:\Merkmale">
        <feature
                name="Merkmal1"
                classificator="classificator1"
                id="merkmal1">
                ...
        </feature>
        ...
</features>
```

Listing 10 zeigt den Bereich für die Konfiguration der Merkmale: *features*.
Durch das Attribut *directory* wird das Verzeichnis festgelegt, worin die Anwendung nach Merkmalsmodulen sucht.
Darin können beliebig viele *feature* Elemente definiert werden. Diese Elemente stellen generische Konfigurationsmöglichkeiten für Merkmalsmodule zur Verfügung. Darin kann beliebiger XML Text deklariert werden, womit die Merkmalsmodule im Detail konfiguriert werden können. Es enthält die Attribute: *name*, *classificator* und *id*.

[29]Technik zur Beschreibung der Struktur von XML Dokumenten.

name definiert den Namen des Merkmals in der Datenbank. Er wird verwendet, um in der Datenbank nach dem Merkmal zu suchen.

classificator gibt den Namen des Klassifikators an, der für das Merkmal verwendet werden soll.

id gibt den Identifizierer des Merkmals an, er muss mit dem Indentifizierer übereinstimmen, der im Modul definiert ist (siehe Abschnitt 5.2.4 unten).

Listing 11: Konfiguration. Timelines.

```
<timelines>
        <timeline name="fussFE"/>
        ...
        <range begin="0" end="30"/>
</timelines>
```

Listing 11 zeigt den Bereich für die Konfiguration der anzuzeigen Zeitreihen von Bewegungsparametern: *timelines*.
Es können beliebig viele *timeline* Elemente definiert werden. Sie beschreiben eine Zeitreihe, die angezeigt wird. Das Attribut *name* gibt den Namen der Zeitreihe in der Datenbank an. Er wird verwendet, um die Zeitreihe aus der Datenbank zu laden.
Die Reihenfolge der Elemente wird auch bei der Anzeige übernommen.
Das Element *range* kann nur einmal definiert werden. Es beschreibt die Breite des Ausschnitts aus den Zeitreihen, der dargestellt werden soll. Wird das Element weggelassen, wird die gesamte Zeitreihe auf einmal dargestellt. Die Attribute *begin* und *end* geben den Anfang und das Ende des Ausschnitts an. Die Einheit dieser Angaben entspricht den konfigurierten Einheiten der Anwendung (siehe das *unitconverter* Element).

Listing 12: Konfiguration. Cutlists.

```
<cutlists default="standardsegmentliste">
        <cutlist name="segmentliste1"/>
        ...
</cutlists>
```

Listing 12 zeigt den Bereich für die Konfiguration von Segmentlisten, die geladen werden sollen: *cutlists*.
Das Attribut *default* gibt die Standardsegmentliste an. Es können beliebig viele *cutlist* Elemente deklariert werden. Sie beschreiben jeweils ein Segmentliste. Das Attribut *name* gibt den Namen der Segmentliste in der Datenbank an. Er wird genutzt, um die Liste in der Datenbank zu finden.

Listing 13: Konfiguration. Unitconverter.

```
<unitconverter accuracy="maximum" units="seconds"/>
```

Listing 13 zeigt den Bereich für die Konfiguration von Einheiten: *unitconverter*. Er enthält die Attribute *accuracy* und *units*.

accuracy gibt die Genauigkeit in Stellen nach dem Komma an, die bei der Umrechnung von Einheiten verwendet werden soll. „maximum" bedeutet, dass die maximale Genauigkeit von 64 Bit Gleitkommazahlen verwendet wird.

units gibt die Einheiten an, die für die Darstellung von Zeitreihen und Segmentlisten verwendet werden. Es können Sekunden, Millisekunden oder Samples (Abtastwerte) sein.

Listing 14: Konfiguration. Classificators.

```
<classificators directory="C:\Classificators">
        <classificator name="classificator1">
                <featureRef id="merkmal1"/>
        </classificator>
</classificators>
```

Listing 14 zeigt den Bereich für die Konfiguration von Klassifikatoren: *classificators*. Das Attribut *directory* gibt das Verzeichnis an, wo die Anwendung nach Klassifikator-Implementierungen suchen soll.
Das Element *classificator* konfiguriert den globalen Klassifikator. Sein Attribut *name* gibt den Namen des Klassifikators an, der als globaler Klassifikator verwendet werden soll. Es kann beliebig viele *featureRef* Elemente enthalten. Sie verweisen auf die Merkmale, die als Eingabedaten für den Klassifikator dienen sollen. Das Attribut *id* gibt den Identifizierer des Merkmals an.

Listing 15: Konfiguration. Videos.

```
<video root="X:\Videos"/>
```

Listing 15 zeigt den Bereich für die Konfiguration von Videoaufnahmen: *videos*. Das Attribut *root* gibt das Wurzelverzeichnis im Dateisystem an, in dem Videoaufnahmen abgelegt sind.

5.2.6. Klassifikatoren

Damit die Visualisierungskomponente eine Einschätzung der Qualität der einzelnen Merkmale sowie die einer gesamten Aufnahme anhand aller Merkmale vornehmen kann, benötigt sie Klassifikatoren. Nach Sklansky ([24, Seite 2]) ist ein Klassifikator ein Gerät oder Prozess, der Daten in Kategorien oder Klassen sortiert. In diesem Kontext handelt es sich um eine Software, die Bewegungsdaten eines Kindes in die Kategorien *Gut* oder *Schlecht* einordnet.

Der Klassifikator, der die Einschätzung der Qualität der gesamten Aufnahme vornimmt, wird in dieser Arbeit **globaler Klassifikator** genannt. Seine Eingabewerte sind alle Merkmalswerte der Aufnahme.

Abbildung 5.16: Definition der Klassifikatorschnittstelle.

Schnittstelle

Die Implementierung der Klassifikatoren selbst ist nicht Teil dieser Arbeit. Dazu existieren viele unterschiedliche Verfahren und es ist Aufgabe der Forschung die passenden Verfahren für diesen Bereich auszuwählen. Es wurde eine Schnittstelle entwickelt, die die Integration von Klassifikatoren in die Anwendung ermöglicht.

Die Schnittstelle wurde als Klassenbibliothek implementiert. Abbildung 5.16 zeigt die Definition der Schnittstelle, die ein Klassifikator implementieren muss. Die Eigenschaft *FalseClassificationRate* liefert die Falschklassifikationsrate des Klassifikators. Sie wird bisher nicht von der Anwendung genutzt. Die Methode *Classify* nimmt einen Datenvektor (dazu später) entgegen und liefert das Klassifikationsergebnis. Das Ergebnis kann Gut, Schlecht oder Unbekannt lauten.

Die Schnittstelle definiert außerdem das Attribut *Classificator*, mit dem der Klassifikator ausgestattet werden muss. Das Attribut registriert den Klassifikator beim DI Container und ermöglicht die Angabe des Klassifikatornamens. Klassifikatoren werden im System durch ihre Namen identifiziert. Die Verknüpfung mit den Modulen erfolgt durch die Angabe ihrer Namen in der Konfiguration.

Datenvektoren können einen oder mehrere Werte enthalten. Ein Datenvektor wird durch die generische Klasse *Vector<T>* definiert, die die nichtgenerische Schnittstelle *IVector* implementiert. Durch diesen Entwurf ist es möglich dieselbe Schnittstelle zu benutzen, um auf Klassifikatoren zuzugreifen, die mit Daten unterschiedlichen Typs arbeiten. Die Klasse ermöglicht den Zugriff auf die Werte über ihren Index oder assoziativ über ihren Namen, wenn den Werten beim Erstellen des Datenvektors Namen zugeordnet wurden.

Klassifikatoren werden in der Anwendung integriert, indem sie als Assemblies erstellt und in das Verzeichnis kopiert werden, das für Klassifikatoren konfiguriert ist. Die Anwendung enthält einen Dienst, *IClassificatorService*, der Klassifikatoren verwaltet und den Zugriff auf sie über ihre Namen ermöglicht.

5.3. Evaluation

Die Evaluation der Visualisierungskomponente wurde durchgeführt, indem die Anwendung einem Arzt am Universitätsklinikum Heidelberg, der mit dem Forschungsprojekt kooperiert, vorgestellt wurde. Danach wurde ein Interview über den Nutzen der Anwendung mit dem Arzt geführt.

Die Vorstellung der Anwendung erwies sich wegen technischen und terminlichen Einschränkungen als schwierig. Deshalb wurde die Vorstellung der Anwendung anhand von Bildschirmfotos, die durch den Arbeitsablauf führen, durchgeführt. Die verwendeten Bildschirmfotos findet der Leser im Anhang (Abschnitt A).

Das Interview mit dem Arzt wurde anhand von vorher festgelegten Fragen geführt. Nachfolgend sind die gestellten Fragen und die Antworten des Arzten aufgeführt.

1. *Sind die Merkmalsgraphen eine sinnvolle Möglichkeit, Fälle miteinander zu vergleichen?*

   Es hängt von den Merkmalen ab. Wenn die Merkmale Auffälligkeiten gut abbilden, sind sie als Vergleichskriterien geeignet. Das ist besonders nützlich, um Fälle zu vergleichen, von denen man nicht weiß dass sie Ähnlichkeiten haben oder um ähnliche Fälle, die man schon kennt, leichter zu finden.

2. *Können Merkmale die Entscheidungsfindung unterstützen?*

   Eventuell ja, wenn weitere aussagekräftige Merkmale gefunden werden, in Verbindung mit dem Merkmalsgraphen, um einen Vergleich zu haben. Wenn die Datenbasis weiter vergrößert wird, kann auch die automatische Klassifikation der Fälle durch das Programm hilfreich bei der Entscheidung sein.

3. *Können Segmentlisten die Entscheidungsfindung unterstützen?*

   Sie sind nützlich, um auffällige Bewegungen innerhalb einer Aufnahme miteinander zu vergleichen.

4. *Können Bewegungsparameter die Entscheidungsfindung unterstütztzen?*

   Unsere bisherige Erfahrung mit Bewegungsparametern (Zeitreihen) hat gezeigt dass es sehr schwierig ist, allein an den Zeitreihen Auffälligkeiten zu erkennen. Nur sehr starke Auffälligkeiten sind deutlich sichtbar. Die Zeitreihen enthalten zuviele unterschiedliche Information über die Bewegung.

5. *Kann die Anwendung zu Schulungszwecken eingesetzt werden?*

   Der Merkmalsgraph kann für Schulungszwecke sehr gut eingesetzt werden. Er kann verwendet werden, um ähnliche Fälle bei der Schulung zu betrachten. Er ist durch die Anzeige des Outcomes und des Benfundes gut geeignet, um sowohl unerfahrenen Ärzten als auch Experten, zu ermöglichen, ihre GM Beurteilung zu überprüfen und zu verbessern.
   Segmentlisten können bei Schulungen sehr gut eingesetzt werden, wenn auffällige Stellen erkannt werden sollen.

6. *Bietet die Anwendung einen Mehrwert im Vergleich zum bestehenden Vorgehen?*

   Ja, sie ergänzt die subjektive Beurteilung des Videos durch objektive Parameter. Sie bietet Vergleichsmöglichkeiten und Unterstützung der Beurteilung.

Fazit

Der entwickelte Ansatz für die Visualisierungskomponente verfügt über gutes Potenzial, um Ärzten Entscheidungsunterstützung zu bieten und die Schulung der Ärzte für die visuelle Analyse der Spontanmotorik zu unterstützen. Wenn die Komponente in Schritt 2. der Beurteilung integriert wird (siehe Abschnitt 2.1), kann eventuell die Sensitivität des Verfahrens gesteigert werden. Dies würde dazu führen, dass mehr betroffene Kinder eine frühe physiotherapeutische Behandlung erfahren könnten, die die Auswirkung von ICP reduzieren kann.

6. Diskussion und Ausblick

In diesem Kapitel werden die beiden Komponenten kritisch betrachtet und ein Ausblick für die weitere Entwicklung gegeben.

6.1. Konfigurationskomponente

Die Konfigurationskomponente hat sich als nützliches Werkzeug für Forscher erwiesen. Allerdings bedarf sie einer Weiterentwicklung, um den Benutzungskomfort zu erhöhen und somit die Akzeptanz zu steigern, sowie um sie an Änderungen der Anforderungen im Zuge der Weiterentwicklung des zugrundeliegenden Softwarepakets anzupassen.

Während der Entwicklung fiel auf, dass es nach der Auslieferung neuer Versionen oder nach Änderungen im Softwarepaket des Forschungsprojekts immer wieder zu Fehlern kam, deren Ursache schwer festzustellen war. Diese Fehler konnten darauf zurückgeführt werden, dass Klassenbibliotheken, von denen die Komponente abhängig ist, in der falschen Version vorlagen oder nicht verfügbar waren.
Bedauerlicherweise treten diese Fehler erst während der Laufzeit auf, generieren keine klaren Fehlermeldungen und können auch nicht durch Protokollierung erfasst werden. Die Ursache dafür liegt in der Abhängigkeit der Komponente von Visual Studio (bzw. der Visual Studio Shell) und deren Bindungsmechanismen für Klassenbibliotheken.
Die Klassenbibliotheken, die während der Laufzeit geladen werden, müssen aus dem Global Assembly Cache[30] (GAC) des .NET Frameworks stammen und können sich von den Bibliotheken unterscheiden, die bei der Entwicklung verwendet wurden.

[30] Ein zentraler Ort des Computers im .NET Framework, in dem gemeinsam genutzte Klassenbibliotheken registriert werden.

Eine Abhilfe für dieses Problem kann ein automatisches Installationsprogramm bieten, das während der Weiterentwicklung der Anwendung mit aktualisiert wird. Es kann dafür Sorge tragen, dass im GAC des Zielsystems die richtigen Bibliotheken zur Verfügung stehen. Ein solches Programm erleichtert auch die Auslieferung erheblich und ist erwünscht, konnte aber im Rahmen dieser Arbeit nicht realisiert werden. Durch die Bindungen an die Visual Studio Shell ist es nicht einfach ein solches Programm zu realisieren.

Weiterhin wäre ein generischer Versionskonverter wünschenswert, der automatisch die benötigten Transformationen zwischen unterschiedlichen Versionen des Konfigurationscripts erkennt und durchführt. Das würde die Notwendigkeit aufheben, bei der Weiterentwicklung der Anwendung mit jeder Version auch einen passenden Versionskonverter für alle vorhergehenden Versionen zu erstellen oder alle bis dato erstellten Konfigurationscripte manuell zu ändern. Ein solches Programm könnte mit der vorhandenen Schnittstelle für Versionskonverter integriert werden.
Die Realisierung könnte durch Verwendung der XSL Transformation[31] in Verbindung mit XML Schemata erfolgen. Bei der Entwicklung der Konfigurationskomponente mit dem VSVM SDK werden automatisch XML Schemata generiert, die die Struktur des Konfigurationscripts und der zugehörigen Layoutdatei beschreiben. Aus den XML Schemata zweier unterschiedlicher Versionen könnte eventuell ein XSL Stylesheet[32] generiert werden, dass die Transformation zwischen den Scripten vornimmt.

Für eine bessere Benutzererfahrung mit der Konfigurationskomponente sollte die Benutzeroberfläche des grafischen Editors verbessert werden.
Es sollten Hilfetexte für jedes Werkzeug und jedes Element mit seinen Eigenschaften angezeigt werden. Das VSVM SDK enthält bereits Möglichkeiten, solche Hilfetexte einzugeben. Es sollten auch Symbole für die grafischen Werkzeuge verwendet werden, die die Funktion der Werkzeuge besser veranschaulichen. Momentan werden Standardsymbole eingesetzt. Der Editor könnte mehr Gebrauch von den, in die Visual Studio Shell integrierten, Fenstern machen, um weitere Bearbeitungsmöglichkeiten bereitzustellen.
Eine hilfreiche Funktion wären außerdem Layoutwerkzeuge, die auf Knopfdruck Elemente auf der Editoroberfläche sinnvoll anordnen können.

6.2. Visualisierungskomponente

Die Visualisierungskomponente hat sich als ein Ansatz mit großem Potenzial erwiesen, Entscheidungsunterstützung und Hilfe bei Schulungen für Ärzte zu bieten.

Die Anwendung basiert auf einer modernen Programmiersprache und Darstellungstechnologie, die von Microsoft gepflegt und weiterentwickelt werden. Daher ist die langfristige Einsetzbarkeit und die Möglichkeit der Weiterentwicklung für die Visualisierungskomponente gegeben.
Die Anwendung bedarf einer Weiterentwicklung um an neue Anforderungen und geänderte Schnittstellen im Zuge der Weiterentwicklung des zugrundeliegenden Softwarepakets

[31]Programmiersprache zur Transformation von XML Dokumenten.
[32]So werden XSLT Programme genannt.

angepasst zu werden. Da die hier vorgestelle Anwendung nur einen ersten Ansatz für die Entwicklung eines entscheidungsunterstützenden Systems darstellt, müssen weitere Konzepte für die Unterstützung der Ärzte einfließen. Besonders Erfahrungen der Ärzte mit der Anwendung müssen in einer Verbesserung und Weiterentwicklung der Anwendung resultieren.

Im Laufe der Entwicklung stellte sich heraus dass die in WPF enthaltene Funktionalität für die Implementierung des Videoplayers nicht geeignet ist. Es wurden mehrere Probleme im Zusammenhang mit dem Einsatz dieser Technik in der Anwendung festgestellt, die z.B. die Implementierung einer geforderten Funktionalität verhinderten, der Möglichkeit die Wiedergabegeschwindigkeit zu ändern. Die Probleme werden in Abschnitt 5.2.3 „Videoplayer“ auf Seite 62 beschrieben.
Eine Implementierung des Videoplayers basierend auf der DirectShow Technologie könnte eine Lösung bieten, da WPF die Integration von DirectShow basierten Elementen erlaubt. Aus Zeitgründen konnte dies in der vorliegenden Arbeit nicht verwirklicht werden.

Die Darstellung der aktuellen Wiedergabeposition in den visualisieren Zeitreihen stellte sich als Performanceproblem heraus. Das Problem wird genauer in Abschnitt 5.2.3 „Timelines“ auf Seite 64 beschrieben. Das Problem konnte teilweise durch stückweises Nachladen der darzustellenden Daten wenn sie gebraucht werden umgangen werden. Die Ursache für das Problem könnte in der Implementierung der Anzeige für die aktuelle Position liegen, die als Erweiterung der *Visiblox* Controls realisiert wurde. Möglicherweise ist die Implementierung nicht optimal oder der Ansatz ungeeignet. Hier müssen weitere Untersuchungen durchgeführt werden.

Ein Problem im Zusammenhang mit der Zoom-Funktion der *Visiblox* Controls fiel während der Entwicklung auf. Das Problem besteht darin, dass der Zoom nach dem schnellen Ausführen mehrerer Zoomvorgänge hintereinander blockiert und es auch nicht zulässt wieder zurück zu zoomen. In der Anwendung tritt das Problem besonders bei der simultanen Anzeige mehrerer Zeitreihen zutage, weil dort die Zoomfunktion für alle Zeitreihen gleichzeitig ausgeführt wird.
Die Ursache für dieses Problem liegt möglicherweise bei WPF. Colin Eberhard beschreibt ein ähnliches Problem und eine Lösungsmöglichkeit in seinem Artikel[33]. Die mögliche Ursache ist, dass das Zeichnen der Benutzeroberfläche nach einem Zoom noch nicht vollständig abgeschlossen ist, während ein erneuter Zoom ausgelöst wird. Hier müssen weitere Untersuchungen durchgeführt werden.

Der Einsatz der kostenlosen Version der *Visiblox* Controls könnte für die Praxis ungeeignet sein, da jedes Control ein Wasserzeichen in seiner Darstellung einbettet. Es sollte daher eine Lizensierung der Controls oder der Umstieg auf eine andere Implementierung erwogen werden.

Ein deklarativer Ansatz bei der Erweiterung der Anwendung um weitere Merkmale kann Forschern die Entwicklung neuer Merkmale erheblich erleichtern. Dabei würde der For-

[33] Colin Eberhard: Throttling Silverlight Mouse Events to Keep the UI Responsive. `http://www.scottlogic.co.uk/blog/colin/2010/06/throttling-silverlight-mouse-events-to-keep-the-ui-responsive/`, letzter Zugriff: 16.04.2011.

scher ein Merkmal in der Konfigurationsdatei eintragen und nur die benötigten Visualisierungsmethoden aus einem Satz von Standardvisualisierungen angeben.
Der bisherige Ansatz wurde gewählt, weil er eine große Flexibilität bei der Entwicklung von Merkmals-Modulen bietet. Das half in der Anfangsphase der Entwicklung als noch nicht klar war, wie die Merkmale sinnvoll visualisiert werden können. Zusätzlich könnte die bisherige Schnittstelle genutzt werden um manuell implementierte Module zu integrieren, die nicht durch den deklarativen Ansatz erstellt werden können.

In der weiteren Entwicklung sollten UnitTests für die Anwendungsmodule implementiert werden. Der modulare Aufbau der Anwendung ermöglicht die einfache Testbarkeit jedes Moduls einzeln. Das würde die Fehlerrate verringern.

Die Anwendung realisiert bisher kein Konzept für Fehlerbehandlung. Es wäre wünschenswert, wenn Fehler in der Anwendung dem Arzt in verständlicher und einheitlicher Form mitgeteilt würden und es eine Möglichkeit gäbe Anwendungsfehler zu korrigieren und mit der Ausführung der Anwendung fortzufahren. Bisher existiert nur eine generische Fehlerbehandlung mit Loggingfunktion.

Um die Benutzerakzeptanz der Anwendung zu verbessern sollte in der weiteren Entwicklung die visuelle Gestaltung der Benutzeroberfläche ansprechender gemacht und das Layout verbessert werden. Es ist auch ein automatisches Installationsprogramm notwendig, dass die Auslieferung der Anwendung erleichtert und zusätzlich die Akzeptanz erhöht.

Literatur

[1] Adde L., Helbostad J.L., Jensenius A.R., Taraldsen G., Støen R.: Using computer-based video analysis in the study of fidgety movements. In: Early Human Development, Jg. 2009, Heft 85/9, S. 541-547.

[2] Berge P.R., Adde L., Espinosa G., Stavdahl Ø.: ENIGMA – Enhanced interactive general movement assessment. In: Expert Systems with Applications, Jg. 2008, Heft 34/4, S. 2664-2672.

[3] Cook S., Jones G., Kent S.: Domain-Specific Development with Visual Studio DSL Tools. Microsoft .Net Development: Addison-Wesley Longman Verlag. Mai 2007.

[4] Einspieler C., Prechtl H.F.R.: Prechtl's assessment of General Movements: A diagnostic tool for the functional assessment of the young nervous system. In: Mental Retardation And Developmental Disabilities Research Reviews, Jg. 2005, Heft 11, S. 61-67.

[5] Ferrari F., Cioni G., Prechtl H.F.R.: Qualitative changes of general movements in preterm infants with brain lesions. In: Early Human Development, Jg. 1990, Heft 23, S. 193–231.

[6] Fowler M.: Presentation Model. `http://martinfowler.com/eaaDev/PresentationModel.html`, Juli 2004, letzter Zugriff: 05.04.2011.

[7] Fowler M.: Inversion of Control Containers and the Dependency Injection pattern. http://martinfowler.com/articles/injection.html#InversionOfControl, Januar 2004, letzter Zugriff: 05.04.2011.

[8] Fowler M.: Event Aggregator. http://martinfowler.com/eaaDev/EventAggregator.html, Septempber 2004, letzter Zugriff: 05.04.2011.

[9] Hadders-Algra M., Groothuis A.M.C.: Quality of general movements in infancy is related to the development of neurological dysfunction, attention deficit hyperactivity disorder and aggressive behavior. In: Developmental Medicine and Child Neurology, Jg. 1999, Heft 41, S. 381–391.

[10] Hadders-Algra M.: General Movements: A window for early identification of children at hight risk for developmental disorders. In: The Journal of Pediatrics, Jg. 2004, Heft 08, S. 512-518.

[11] Heinze F., Breitbach-Faller N., Schmitz-Rode1 T., Disselhorst-Klug C.: Movement Analysis by Accelerometry of Newborns for the Early Detection of Movement Disorders due to Infantile Cerebral Palsy. In: IFMBE Proceedings, Jg. 2009, Heft 25, S. 24-27.

[12] Huber S.: Entwicklung eines grafischen Editors zur Erstellung zustandsorientierter multizellulärer Gewebemodelle. Diplomarbeit, Universität Heidelberg, 2006.

[13] Karch D.: Quantitative Analyse der Spontanmotorik von Säuglingen für die Prognose der infantilen Cerebralparese. Dissertation, Universität Heidelberg, 2011.

[14] Lasser D.J.: Topological ordering of a list of randomly-numbered elements of a network. In: Communications of the ACM, Jg. 1961, Heft 4.

[15] MacDonald M.: Pro WPF in C# 2010: Windows Presentation Foundation in .Net 4. Expert's Voice in .NET: Apress, März 2010.

[16] Meinecke L., Breitbach-Faller N., Bartz C., Damen R., Rau G., Disselhorst-Klug C.: Movement analysis in the early detection of newborns at risk for developing spasticity due to infantile cerebral palsy. In: Human Movement Science, Jg. 2006, Heft 25, S. 125-144.

[17] Projektseite von MEF auf Codeplex.de. Learn more about MEF. http://mef.codeplex.com/wikipage?title=Overview&referringTitle=Home, letzter Zugriff: 05.04.2011

[18] Microsoft Dokumentation zu VSVM. http://msdn.microsoft.com/en-us/library/bb126259%28VS.100%29.aspx, letzter Zugriff: 05.04.2011.

[19] Microsoft Prism 4 User Guide. http://msdn.microsoft.com/en-us/library/gg406140.aspx, November 2010, letzter Zugriff: 05.04.2011.

[20] Prechtl H.F.R.: Qualitative changes of spontaneous movements in fetus and preterm infant are a marker of neurological dysfunction. In: Early Human Development, Jg. 1990, Heft 23, S. 151-158.

[21] Prechtl H.F.R., Ferrari F, Cioni G.: Predictive value of general movements in asphyxiated fullterm infants. In: Early Human Development, Jg. 1993, Heft. 35, S. 91–120.

[22] Recchia P.: Provide an improved user experience to your DSL's. `http://www.netfxfactory.org/blogs/papers/archive/2009/09/20/provide-an-improved-user-experience-to-your-dsls.aspx`, September 2009, letzter Zugriff: 07.04.2011.

[23] Schneider H.: Lexikon der Informatik und Datenverarbeitung: 4. Auflage, R. Oldenbourg Verlag München Wien, 1997.

[24] Sklansky J., Wassel G.N.: Pattern Classifiers and Trainable Machines: Springer-Verlag, 1981.

[25] Sølvberg A., Kung D.C.: Information Systems Engineering. An Introduction: Springer-Verlag, 1993.

[26] Stahl T., Völter M., Efftinge S., Haase A.: Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management: 2. Auflage, d.punkt Verlag, Mai 2007.

[27] Vogel O., Arnold I., Chughtai A., Ihler E., Kehrer T., Mehlig U., Zdun U.: Software-Architektur. Grundlagen-Konzepte-Praxis: 2. Auflage, Spektrum Akademischer Verlag, 2009.

Glossar

.NET Softwareplattform von Microsoft. Sie enthält eine Laufzeitumgebung zur Ausführung von Programmen und eine Klassenbibliothek, die wiederverwendbare Typen enthält und Lösungen für viele gänginge Programmieraufgaben bietet.

Assembly In .NET wird ein übersetztes Programm als assembly bereitgestellt. Das ist die ausführbare Sammlung der Klassen des Programms mit zusätzlichen Metadaten.

Attribute Name für ein Konstrukt in .NET, um Programmkomponenten (Klassen, Methoden, ...) deklarativ mit Metadaten zu assoziieren. Diese Technik existiert auch in anderen objektorientierten Programmiersprachen unter anderen Namen.

Bewegungsparameter Quantitative Parameter der kindlichen Bewegung, die mithilfe von technischen Hilfsmitteln erfasst und berechnet werden.

Datentyp Bezeichnet bestimmte Arten von Objekten mit darauf definierten Operationen bei typisierten Programmiersprachen.

Eigenschaften In .NET sind Eigenschaften (oder Properties) Accessoren, die den Zugriff auf Datenfelder von Klassen kapseln und Sicherheitsüberprüfungen und Berechnungscode enthalten können.

Entwurfsmuster Lösungsschablonen für wiederkehrende Entwurfsprobleme in der objektorientierten Softwarearchitektur.

GAC Global Assembly Cache in .NET. Ein System zur Registrierung von Assemblies an einem zentralen Ort des Computers. Erlaubt die gemeinsame Nutzung von Assemblies durch unterschiedliche Programme, die in der .NET Laufzeitumgebung ausgeführt werden.

Generische Typparameter Ein Konstrukt in .NET, um Klassen zu definieren, die mit Werten unterschiedlicher Typen arbeiten können. Um eine solche Klasse zu verwenden müssen ihre Typparameter mit konkreten Typen gefüllt werden.

ICP Abkürzung für Infantile Cerebralparese. Bezeichnet in der Medizin eine Störung des Muskel- und Nervensystems, hervorgerufen durch eine frühkindliche Hirnschädigung.

Kardinalitäten In der Datenmodellierung, die Anzahl der Elemente, mit denen ein Element in Beziehung stehen kann oder muss.

Konfigurationscript Enthält eine Reihe von Anweisungen, die die Einstellungen einer Anwendung setzen.

Merkmale Quantitative Größen, die aus Bewegungsparametern errechnet werden und eine Unterscheidung bezüglich normalen und abnormalen Bewegungen im Sinne der General Movements Analyse ermöglichen.

Metadaten Metadaten sind Daten, die Informationen über andere Daten enthalten. In .NET werden Informationen Programmcode als Metadaten des Programms bezeichnet.

Namespace Zu Deutsch Namensraum. In der objektorientierten Programmierung wird durch namespaces eine hierarchische Struktur definiert, in die Klassen eingeordnet werden. Durch den namespace und den Klassennamen können Klassen eindeutig identifiziert werden.

Reflection In .NET, eine Technik, die es erlaubt Informationen über Datentypen zur Laufzeit auszulesen.

Schnittstelle In .NET definiert eine Schnittstelle einen Typ aber bietet keine Implementierung seiner Funktionalität an. Somit ist die Schnittstelle in .NET mit einer abstrakten Klasse in anderen Programmiersprachen vergleichbar. Schnittstellen können in .NET als einzige Typen mehrfach geerbt werden.

Serialisierung In der Informatik ist es die Abbildung von Objekten auf eine externe Darstellungsform.

UserControl In WPF ist ein UserControl eine benutzerdefinierte grafische Komponente der Benutzerschnittstelle, die sich aus unterschiedlichen Komponenten zusammensetzt Diese Zusammensetzung kann eine eigene Schnittstelle definieren und in WPF als eigenständige Komponente benutzt werden.

XML Abkürzung für Extensible Markup Language. Weit verbreitete Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Textform. Wird vom World Wide Web Consortium spezifiziert.

XML Schema Eine vom World Wide Web Consortium empfohlene Technik zur Beschreibung der Struktur von XML Dokumenten. XML Schemata werden von vielen XML Editoren unterstützt und ermöglichen die Validierung von XML Dokumenten.

XSL Transformation Abgekürzt XSLT ist eine Programmiersprache zur Transformation von XML Dokumenten. Sie wird selbst als XML Dokument aufgebaut und von sogenannten XSLT Prozessoren verarbeitet.

Abbildungsverzeichnis

A. Benutzerhandbuch Visualisierungskomponente

1 Einleitung

Die Visualisierungskomponente ist ein Schritt in Richtung entscheidungsunterstützendes System. Sie visualisiert objektive Merkmale und Bewegungsparameter der Spontanmotorik für den beurteilenden Arzt und bietet eine Möglichkeit, unterschiedliche Fälle anhand der objektiven Merkmale miteinander vergleichen zu können.

1.1 Überblick

Das Anwendungsfenster enthält sieben logische Bereiche. Abbildung 1 zeigt das Anwendungsfenster nach dem Programmstart.
Die Werkzeugleiste ist der obere Bereich. Sie enthält Schaltflächen, mit denen allgemeine Funktionen der Anwendung aktiviert und das Aussehen bestimmt werden kann.
Die linke Seite des Anwendungsfensters wird vom Detailbereich eingenommen. Die rechte Seite enthält fünf untereinander angeordnete Bereiche. Auf den Detailbereich und die anderen fünf Bereiche wird später eingegangen.

1.2 Anwendungsstart

Die Anwendung stellt nach dem Start eine Verbindung zu den Datenbanken her. Sie werden bevor die Anwendung geladen ist, nach dem Benutzernamen und Passwort für die Datenbankverbindung gefragt (Abbildung 2). Im Auswahlfeld „Benutzer" werden ihnen Benutzernamen als Auswahlmöglichkeiten angezeigt, die für ihre Programmversion vorkonfiguriert sind.

2 Arbeiten mit der Visualisierungskomponente

In den Folgenden Abschnitten wird Beschrieben wie Sie die Schritte in einem typischen Arbeitsablauf mit der Visualisierungskomponente durchführen.

2.1 Daten laden

Klicken Sie auf „Open Child" in der Werkzeugleiste. Abbildung 3 zeigt das sich öffnende Fenster. Auf der linken Seite des Fensters sehen Sie die zur Auswahl stehenden Kinder. Das Eingabefeld darüber erlaubt es, die Liste zu filtern. Geben Sie hier den Anfang des Vor- oder Nachnamens des gesuchten Kindes ein (**Achtung**: Groß- und Kleinschreibung beachten). Die Liste der Kinder wird automatisch aktualisiert.

Wählen Sie ein Kind aus, um auf der rechten Seite die Aufnahmen zu sehen, die zu diesem Kind existieren (Abbildung 4). Die Einträge bestehen aus dem Aufnahmedatum. Wählen Sie die gewünschte Aufnahme aus und bestätigen Sie mit „OK".

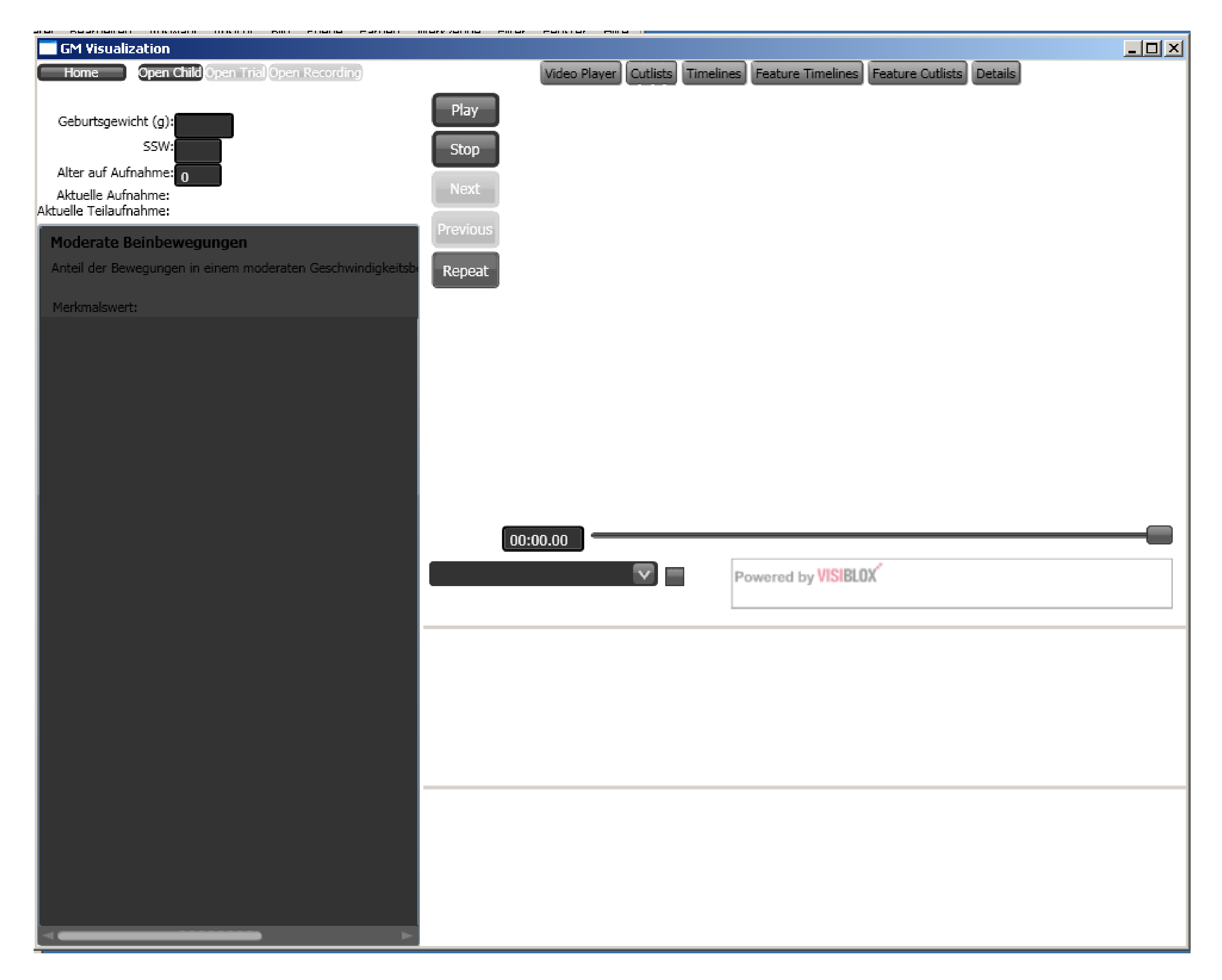


Abbildung 1: Hauptansicht der Anwendung nach dem Programmstart.

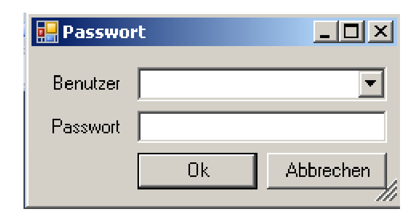


Abbildung 2: Abfrage der Zugangsdaten für die Datenbankverbindung.

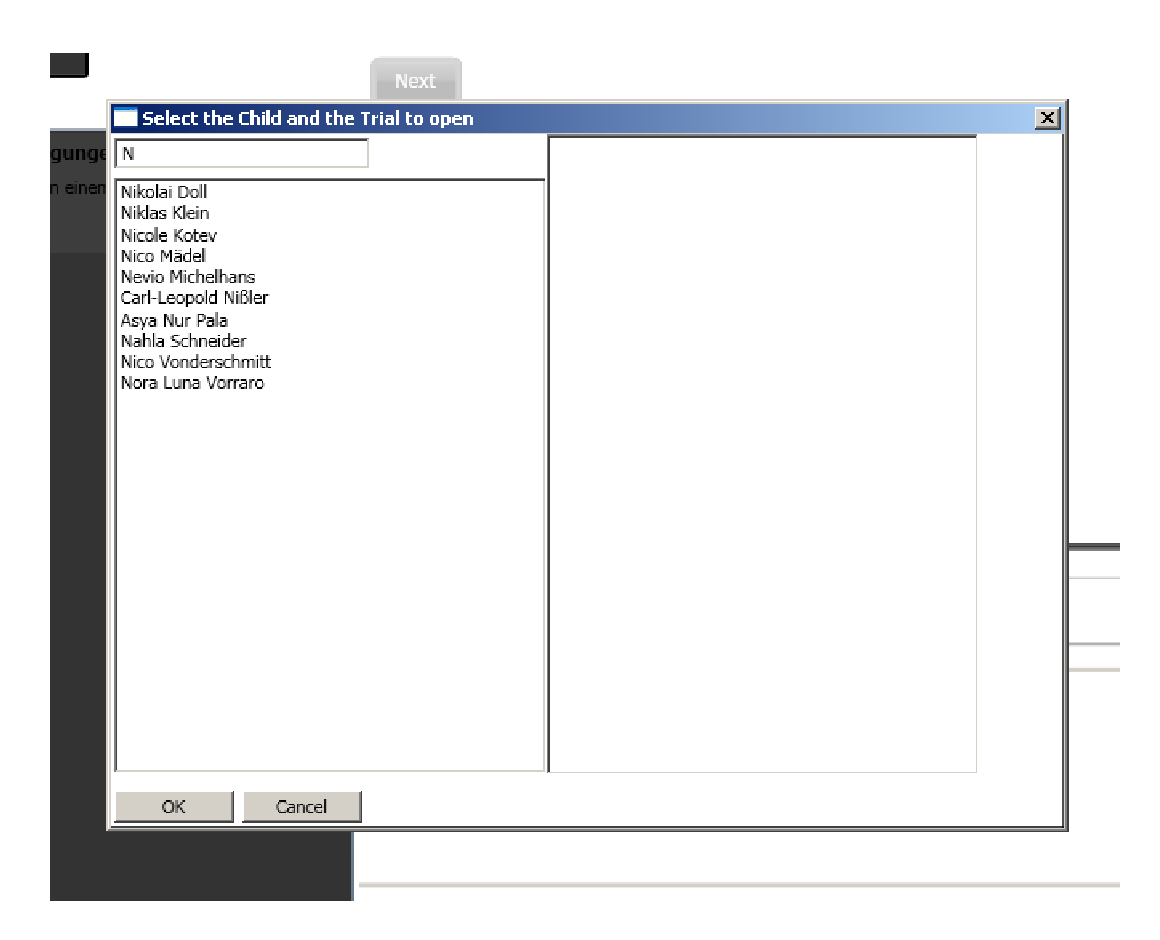


Abbildung 3: Auswahlfenster beim Laden eines Falls.

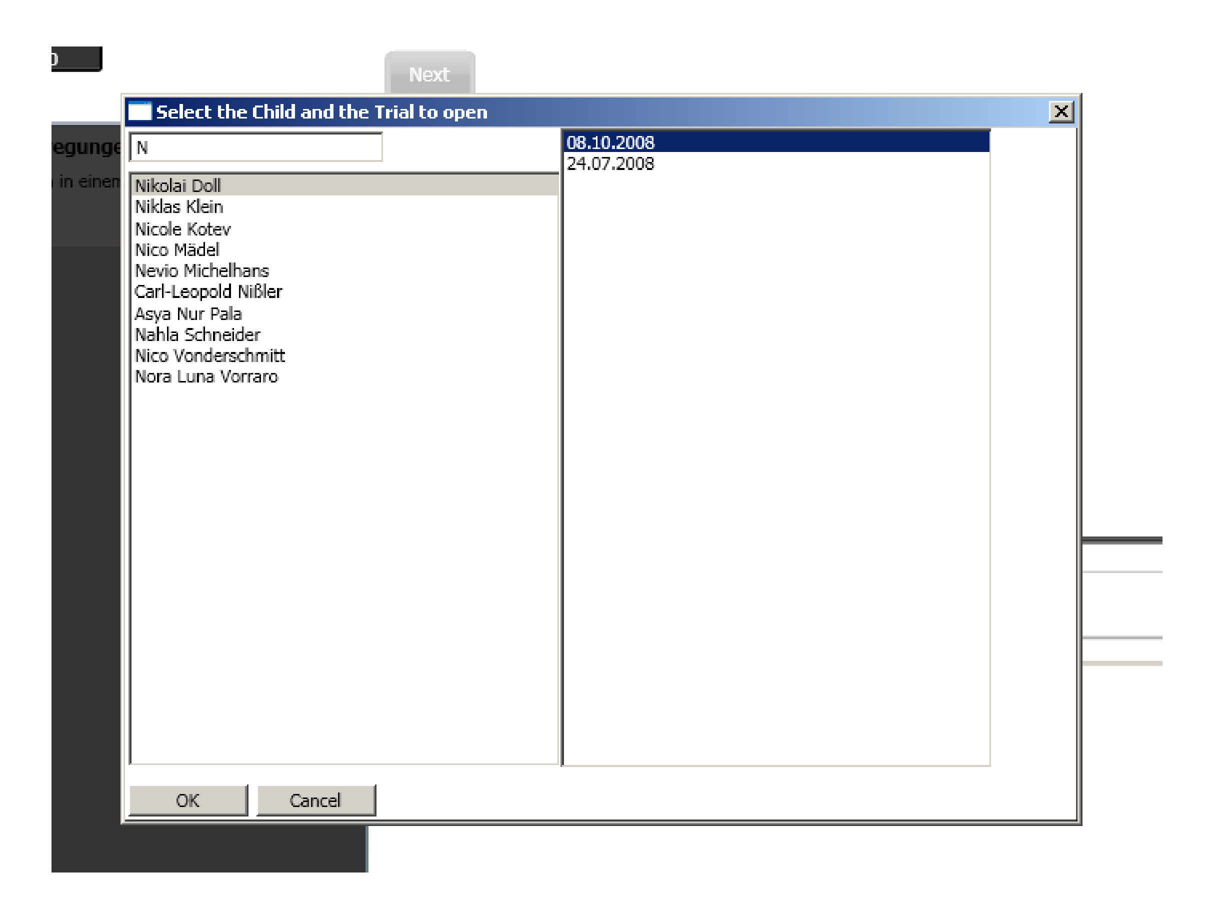


Abbildung 4: Kind im Auswahlfenster selektiert.

Wenn keine Standardaufnahme konfiguriert ist, erscheint in diesem Fenster noch ein dritter Bereich, wo sie die Teilaufnahme wählen können. Wenn dieser Bereich nicht erscheint, wird die konfigurierte Standardteilaufnahme geladen (soweit für den gewählten Fall vorhanden). (Siehe Abschnitt 3)

Die Daten des gewählten Falls und der Aufnahme werden geladen und angezeigt. Abbildung 5 zeigt das Anwendungsfenster mit geladenen Daten.

Momentan erscheint das Videobild erst, wenn der Abspielvorgang einmal gestartet worden ist (genauso erscheint beim laden eines anderen Falls das neue Videobild erst, wenn der Abspielvorgang gestartet wird). Möglicherweise wird die Größe des Videoplayers nach dem Starten der Wiedergabe durch das Video angepasst. Sie können sie dann wieder so wie gewünscht einstellen (Siehe Abschnitte 2.4 und 2.2.2).

Die Schaltflächen „Open Trial" und „Open Recording" sind jetzt aktiv und sie können eine andere Aufnahme oder Teilaufnahme laden.
Klicken Sie auf „Open Trial", um eine andere Aufnahme zu laden. Abbildung 6 zeigt das Fenster das sich daraufhin öffnet. Wählen Sie das Datum der Aufnahme und bestätigen Sie mit „OK".

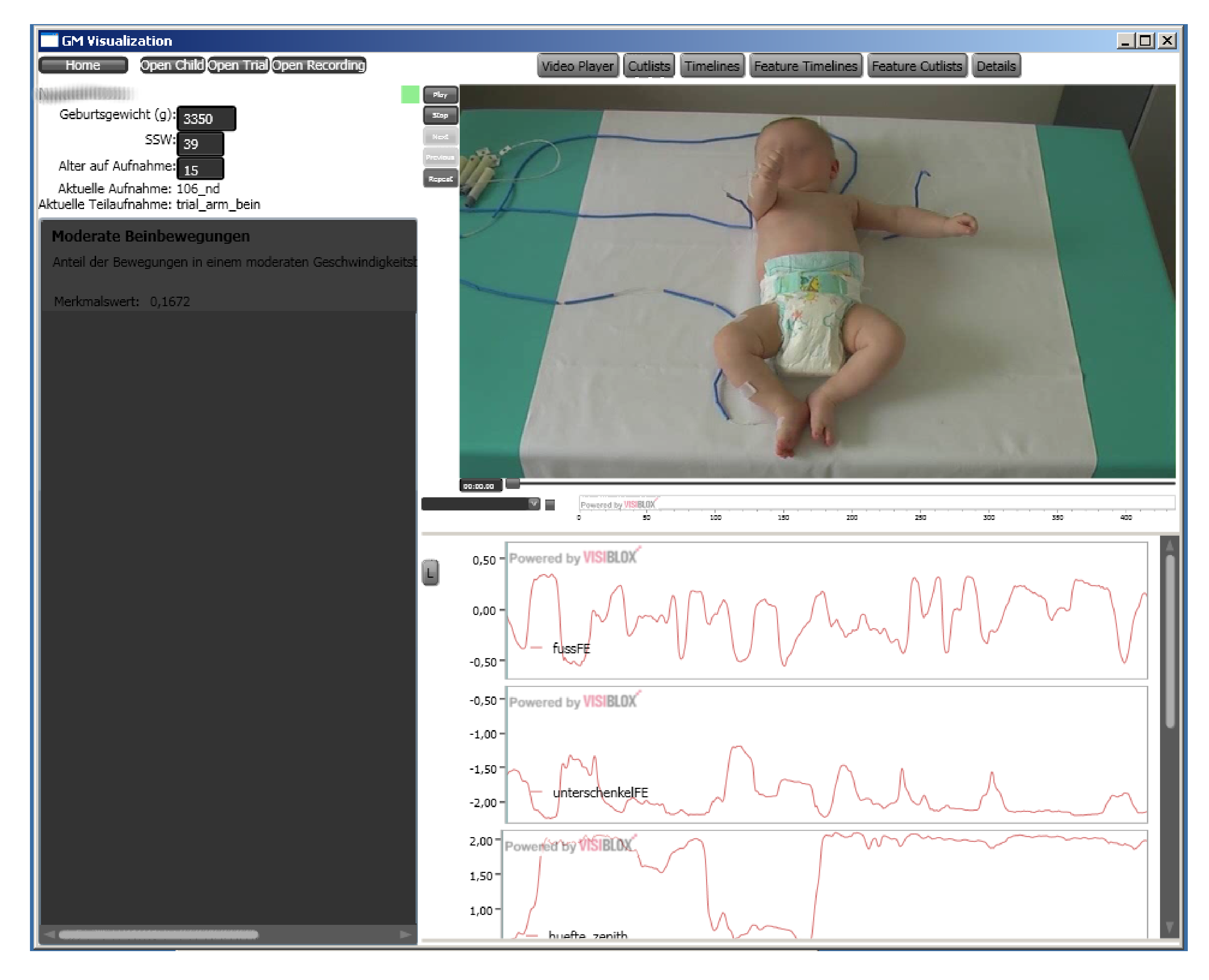


Abbildung 5: Hauptansicht der Anwendung wenn ein Fall geladen wurde.

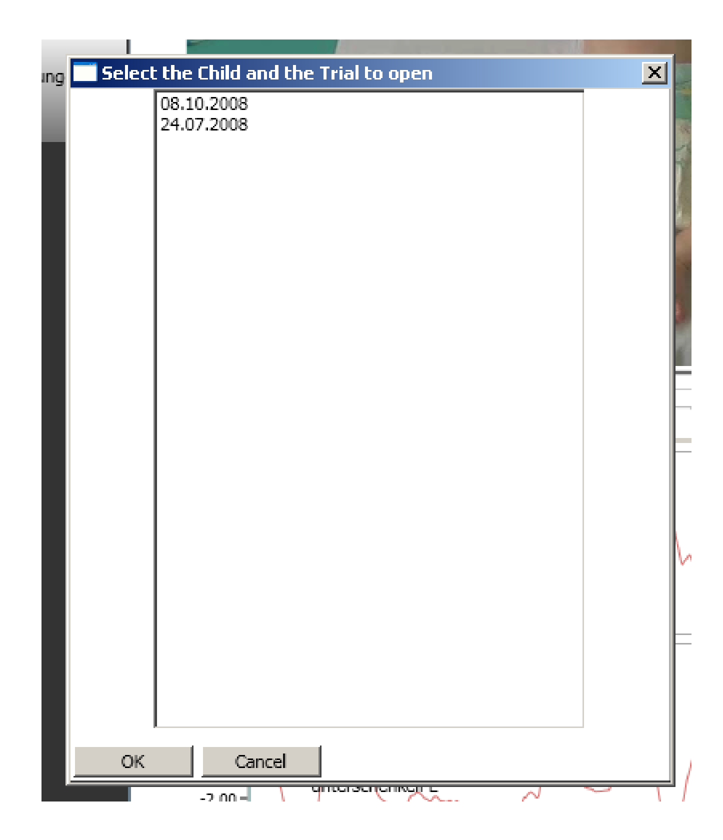


Abbildung 6: Auswahlfenster beim Laden einer Aufnahme.

Klicken Sie auf „Open Recording", um eine andere Teilaufnahme zu laden. Abbildung 7 zeigt das sich öffnende Fenster. Wählen Sie den Namen der Teilaufnahme und bestätigen Sie mit „OK".

2.2 Bereiche der Hauptansicht

In den folgenden Abschnitten werden die verschiedenen Bereiche in der Hauptansicht der Anwendung beschrieben.

2.2.1 Detailbereich

Oben links werden Detailinformationen zum geladenen Fall präsentiert.

Abbildung 8 zeigt den Bereich mit folgenden Daten:

- Der Vor- und Nachname des Kindes
- Die Schwangerschaftswoche bei der Entbindung
- Das Alter des Kindes auf der gewählten Aufnahme in Wochen (unkorrigierte Angabe ab dem tatsächlichen Geburtszeitpunkt)

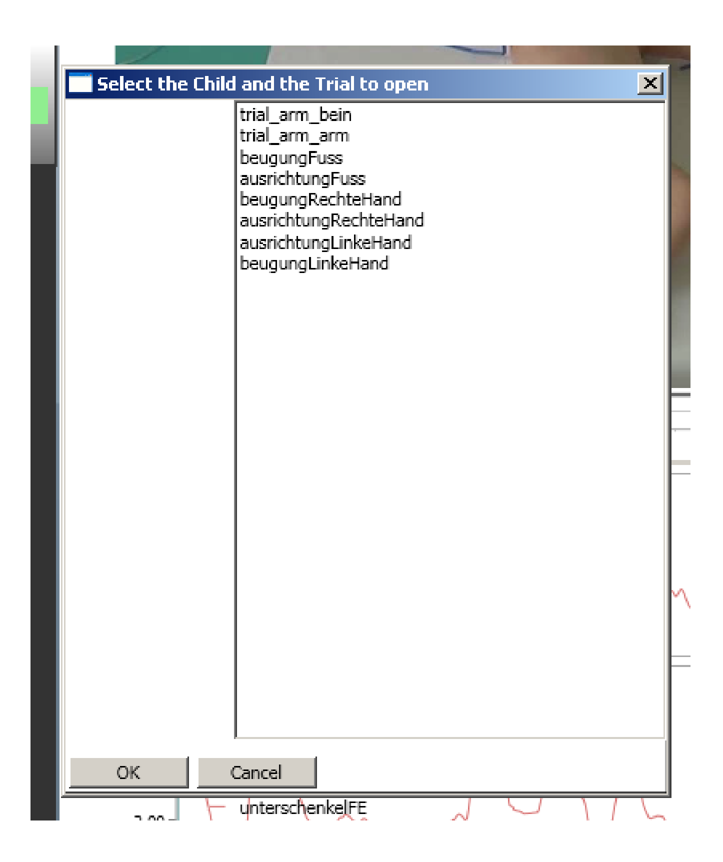


Abbildung 7: Auswahlfenster beim Laden einer Teilaufnahme.

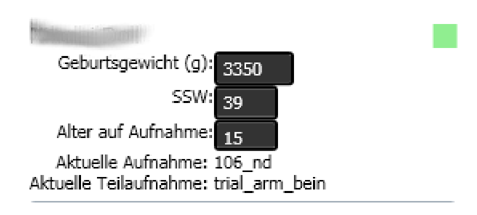


Abbildung 8: Anzeige der Detailinformationen zum Fall.

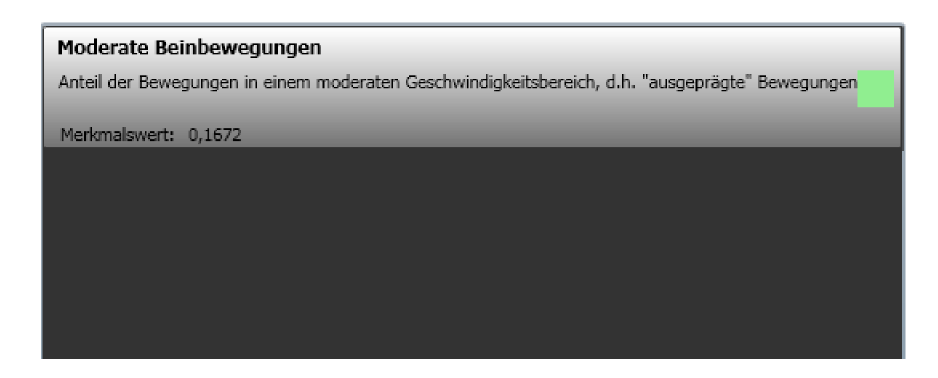


Abbildung 9: Auflistung von Merkmalen.

- Name der geladenen Aufnahme
- Name der geladenen Teilaufnahme

Außerdem wird eine Einschätzung der Bewegungsqualität durch die Anwendung angezeigt. Sie wird durch ein Rechteck in der rechten oberen Ecke dieses Bereichs dargestellt. Ist das Rechteck grün, wurde die Bewegungsqualität als gut eingestuft, ist es rot, als schlecht. Wenn das Rechteck nicht sichtbar ist, wurde keine Einschätzung durchgeführt (z.B. aus Mangel an Daten).

Darunter sehen Sie eine Auflistung von Merkmalen. Die Elemente der Auflistung zeigen alle verfügbaren Merkmale mit ihrem Namen und ihrer Beschreibung (Abbildung 9).

Die Elemente der Auflistung enthalten außerdem die Werte der Merkmale für die geladene Aufnahme (falls verfügbar) und eine Einschätzung der Qualität der einzelnen Merkmale durch die Anwendung. Sie wird analog zu der Einschätzung der Bewegungsqualität präsentiert.

2.2.2 Videoplayer

Oben rechts befindet sich der Videoplayer (Abbildung 10). Er hat folgende Bedienelemente:

- Schaltfläche zum Abspielen / Pausieren der Videoaufnahme
- Schaltfläche zum Stoppen der Wiedergabe
- Schaltflächen um zum nächsten und vorherigen Segment zu springen
- Schaltfläche um Wiederholung der Wiedergabe zu aktivieren oder zu deaktivieren
- Schieberegler mit dem die Abpielposition verändert werden kann
- Eine Anzeige unten links zeigt die aktuelle Wiedergabezeit im Format: *Minuten:Sekunden.Hundertstels* an

Abbildung 10: Videoplayer.

Das Seitenverhältnis der angezeigten Videos beträgt 16 zu 9. Die Größe des Videoplayers passt sich automatisch dem Video an.

Sollte der Videoplayer einmal nicht mehr richtig reagieren, versuchen Sie die Aufnahme zu stoppen und wieder zu starten.

2.2.3 Segmentliste

Unter dem Videoplayer befindet sich die Segmentliste (siehe Abbildung 11). Segmentlisten markieren Abschnitte der Aufnahme und enthalten Informationen über sie.
Die Segmente werden als gelbe Rechtecke dargestellt. Die Skala unter der Darstellung der Segmente ist die Zeit in der Aufnahme. Links von den Segmenten befindet sich ein Auswahlfeld, dass die Namen der verfügbaren Segmentlisten enthält. Wenn Sie eine Segmentliste auswählen wird sie geladen und rechts dargestellt.
Das Kontrollkästchen rechts von dem Auswahlfeld aktiviert oder deaktiviert das Abspielen der geladenen Segmentliste. Wird eine Segmentliste aktiviert, können nur noch die Segmente abgespielt werden. Mit den Schaltflächen „Next" und „Previous" des Videoplayers können Segmente umgeschaltet werden. Segmente können auch direkt abgespielt werden, wenn Sie sie in der Segmentliste anklicken.

Der Wiederholmodus des Videoplayers wiederholt immer nur ein Segment ab. Sie können aber jederzeit zu einem anderen Segment springen.

CutSegment Powered by VISIBLOX 0 50 100 150 200 250 300 350 400

Abbildung 11: Segmentliste.

Wenn eine Segmentliste aktiv ist, werden Informationen über das gerade abgespielte Segment im Videobild eingeblendet. Wenn keine Segmentliste aktiv ist, werden Informationen über Segmente der zuletzt geladenen *Segmentliste angezeigt, wenn die Wiedergabeposition ein Segment erreicht.*

2.2.4 Zeitreihen

Unter der Segmentliste befinden sich die Zeitreihen (siehe Abbildung 12). Zeitreihen visualisieren Bewegungsparameter der geladenen Aufnahme.
Es werden alle in der Konfiguration angegebenen Zeitreihen angezeigt sofern sie für die geladene Aufnahme verfügbar sind (Siehe Abschnitt 3). Links von den Zeitreihen befindet sich die Werteskala. Unter der letzten Zeitreihe befindet sich die Zeitskala. Der Name der Zeitreihe befindet sich innerhalb der Visualisierung auf der linken Seite. Der Bereich der Zeitreihe der angezeigt wird ist in der Konfiguration angegeben. Überschreitet die aktuelle Wiedergabeposition den angezeigten Bereich, wird er entsprechend verschoben. Die aktuelle Wiedergabeposition wird in den Zeitreihen durch die blaue Linie angezeigt. Sie können diese Linie verschieben, um an eine andere Stelle der Wiedergabe zu springen. Mit der Schaltfläche links von den Zeitreihen (L) kann die Anzeige der aktuellen Position in den Zeitreihen an- und ausgeschaltet werden.

Die Anzeige der Zeitreihen hat eine Zoomfunktion. Um einen Bereich zu vergrößern, klicken Sie auf eine Stelle in der Zeitreihe und ziehen Sie mit gedrückter Maustaste ein blaues Rechteck um den Bereich, den Sie vergrößern möchten (In Abbildung 12 ist ein Zoom-Rechteck dargestellt). Wenn Sie die Maustaste loslassen, wird der gewählte Bereich in allen Zeitreihen vergrößert. Um die Darstellung zurückzusetzen, doppelklicken Sie auf die Zeitreihe.

Es kann passieren, dass der Zoom plötzlich nicht mehr funktioniert. Probieren sie in diesem Fall in einer anderen Zeitreihe zu zoomen.

Bei simultaner Anzeige vieler Zeitreihen oder der Anzeige von breiten Bereichen kann eine verlangsamte oder abgehackte Videowiedergabe auftreten. In diesem Fall können Sie die Anzeige der aktuellen Wiedergabeposition abschalten, um die Performance zu verbessern.

2.3 Bereiche der Merkmalsansicht

Die Merkmalsansicht zeigt die Details zu einem ausgewählten Merkmal. Sie gelangen zu dieser Ansicht, wenn sie in der Hauptansicht auf ein Merkmal in der Auflistung doppelklicken, die sich im Detailbereich befindet. Um wieder zur Hauptansicht zu gelangen,

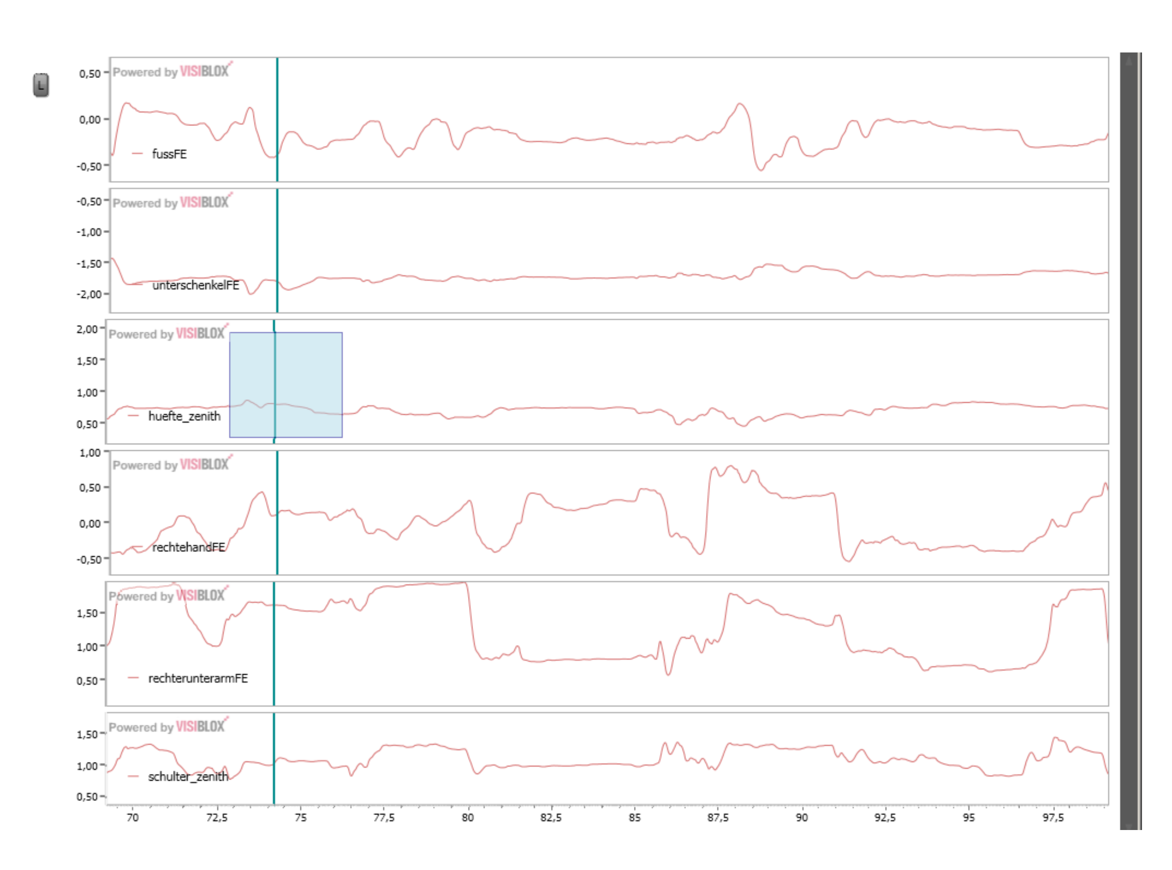


Abbildung 12: Zeitreihen.

klicken Sie auf die Schaltfläche „Home" in der Werkzeugleiste. Abbildung 13 zeigt die Merkmalsansicht für das Merkmal „Moderate Beinbewegungen" mit geladenen Daten.

Merkmale können als Merkmal-Module in die Anwendung integriert werden. Die im Folgenden beschriebenen Visualisierungen sind die Standardvisualisierungen von Merkmal-Modulen. Allerdings kann ein Hersteller eines Merkmal-Moduls die Visualisierung des Merkmals beliebig anpassen. Wenn die Darstellung eines der Merkmale von der hier beschriebenen abweicht, konsultieren Sie den Hersteller des Merkmal-Moduls.

Im folgenden werden die Bereiche dieser Ansicht beschrieben die sich von denen der Hauptansicht unterscheiden.

2.3.1 Detailbereich

Links oben wird der Name des Merkmals und die Einschätzung seiner Qualität angezeigt (Abbildung 14).

Darunter befindet sich der Merkmalsgraph und Detailinformationen zum Merkmal (Abbildung 15).
Der Merkmalsgraph stellt die Werte des selektierten Merkmals aller vergleichbarer Fälle aus der Datenbank und des geladenen Falls dar. Unter dem Graph befindet sich eine Legende, die die verschiedenen Symbole im Graph erklärt. Der Graph ist eindimensional, da das Merkmal nur einen Wert aufweist, die Einteilung auf der y-Achse dient nur besserer Übersicht.
Durch Anklicken eines Symbols im Graphen kann der entsprechende Fall geladen werden. Abbildung 16 zeigt das Bestätigungsfenster das daraufhin angezeigt wird.
Die Detailinformationen können verschiedenartig gestaltet sein. Zum Beispiel können sie Kennzahlen des Merkmals oder eine textuelle Beschreibung der Merkmalsqualität enthalten. In Abbildung 15 wird eine Beschreibung des Merkmals in der Detailansicht angezeigt.

2.3.2 Merkmals-Segmentlisten

Auf der Rechten Seite befinden sich unter dem Bereich *Segmentlisten* die Merkmals-Segmentlisten (Abbildung 17). Sie stellen Segmentlisten dar die durch das Merkmal erzeugt wurden. Sie funktionieren analog den *Segmentlisten* außer dass hier mehrere Segmentlisten gleichzeitig angezeigt werden.
Die Anwendung erlaubt nur eine gleichzeitig aktive Segmentliste. Wird eine andere Segmentliste aktiviert, wird die vorher aktivierte automatisch deaktiviert.
Welche Segmentlisten angezeigt werden, wird unabhängig von den *Segmentlisten* (Abschnitt 2.2.3) in der Konfiguration festgelegt (Siehe Abschnitt 3).

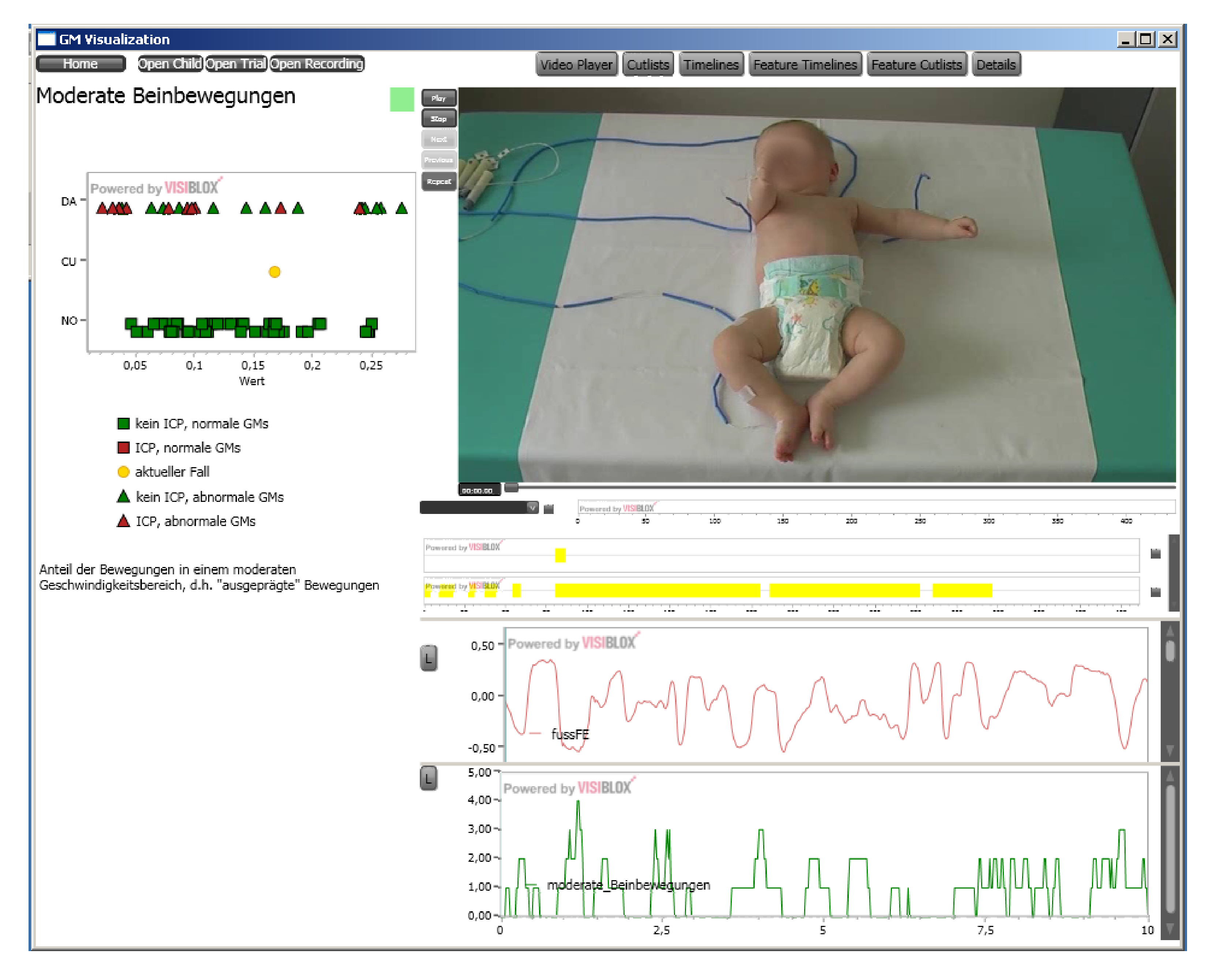


Abbildung 13: Merkmalsansicht der Anwendung mit geladenen Daten.

Moderate Beinbewegungen

Abbildung 14: Name und Qualität des Merkmals.

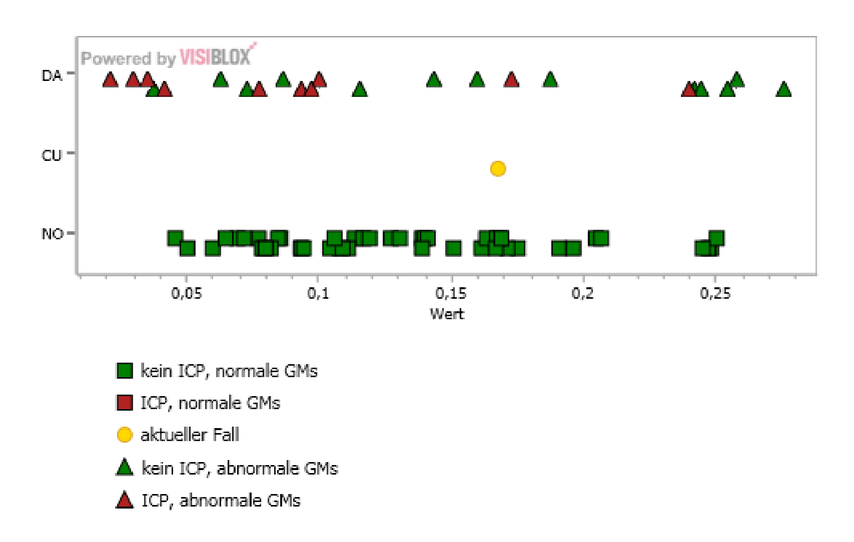


Abbildung 15: Merkmalsgraph und Detailinformationen des Merkmals „Moderate Beinbewegungen”.

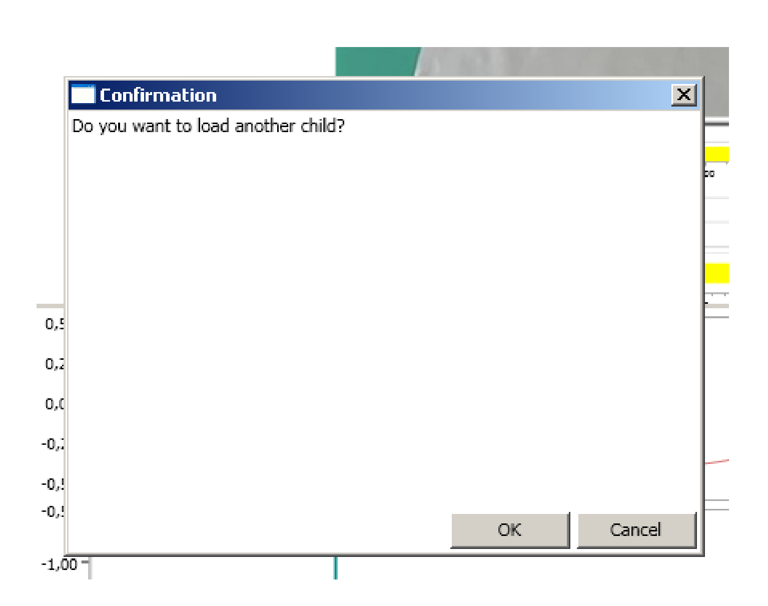


Abbildung 16: Bestätigungsfenster beim Laden eines Falls aus dem Merkmalsgraph.

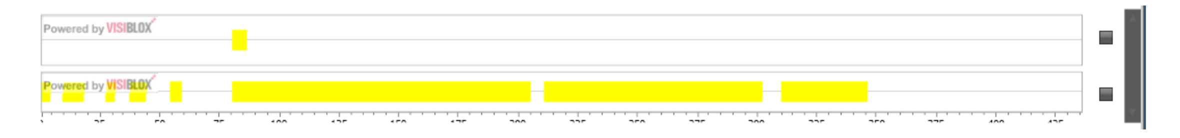


Abbildung 17: Merkmals-Segmentlisten.

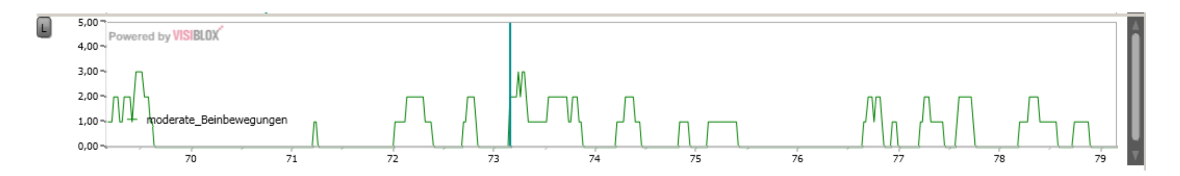


Abbildung 18: Merkmals-Zeitreihen.

2.3.3 Merkmals-Zeitreihen

Unterhalb der *Zeitreihen* auf der rechten Seite befinden sich die Merkmals-Zeitreihen (Abbildung 18). Sie stellen Zeitreihen dar, die durch das Merkmal erzeugt werden. Sie funktionieren analog zu den *Zeitreihen*.
Welche Zeitreihen angezeigt werden, wird unabhängig von den *Zeitreihen* (Abschnitt 2.2.4) in der Konfiguration festgelegt (Siehe Abschnitt 3).

2.4 Anpassen des Layouts

Das Layout der Anwendung kann durch verschiedene Werkzeuge angepasst werden.
Die Breite des Detailbereichs ändert sich wenn die Fenstergröße geändert wird, nimmt aber maximal ein Drittel der Fensterbreite ein.
Auf der rechten Fensterseite befinden sich zwei horizontale graue Linien, die das Fenster in drei Zeilen teilen. Sie können nach oben oder unten bewegt werden und vergrößern oder verkleinern die Bereiche, die um sie herum liegen. Die obere graue Linie teilt den Raum zwischen den Zeitreihen und dem Videoplayer mit Segmentlisten und Merkmals-Segmentlisten ein. Die untere graue Linie teilt den Raum zwischen Zeitreihen und Merkmals-Zeitreihen ein.
In der Werkzeugleiste befinden sich Schaltflächen zum Ein- und Ausblenden von Bereichen. Abbildungen 19 bis 23 demonstrieren die Anwendung mit verschiedenen ausgeblendeten Bereichen.

Momentan werden Layouteinstellungen nicht gesichert. D.h. sie müssen beim nächsten Programmstart erneut angepasst werden.

Abbildung 19: Ausgeblendeter Detailbereich.

Abbildung 20: Ausgeblendete Detailbereich und Merkmals-Segmente.

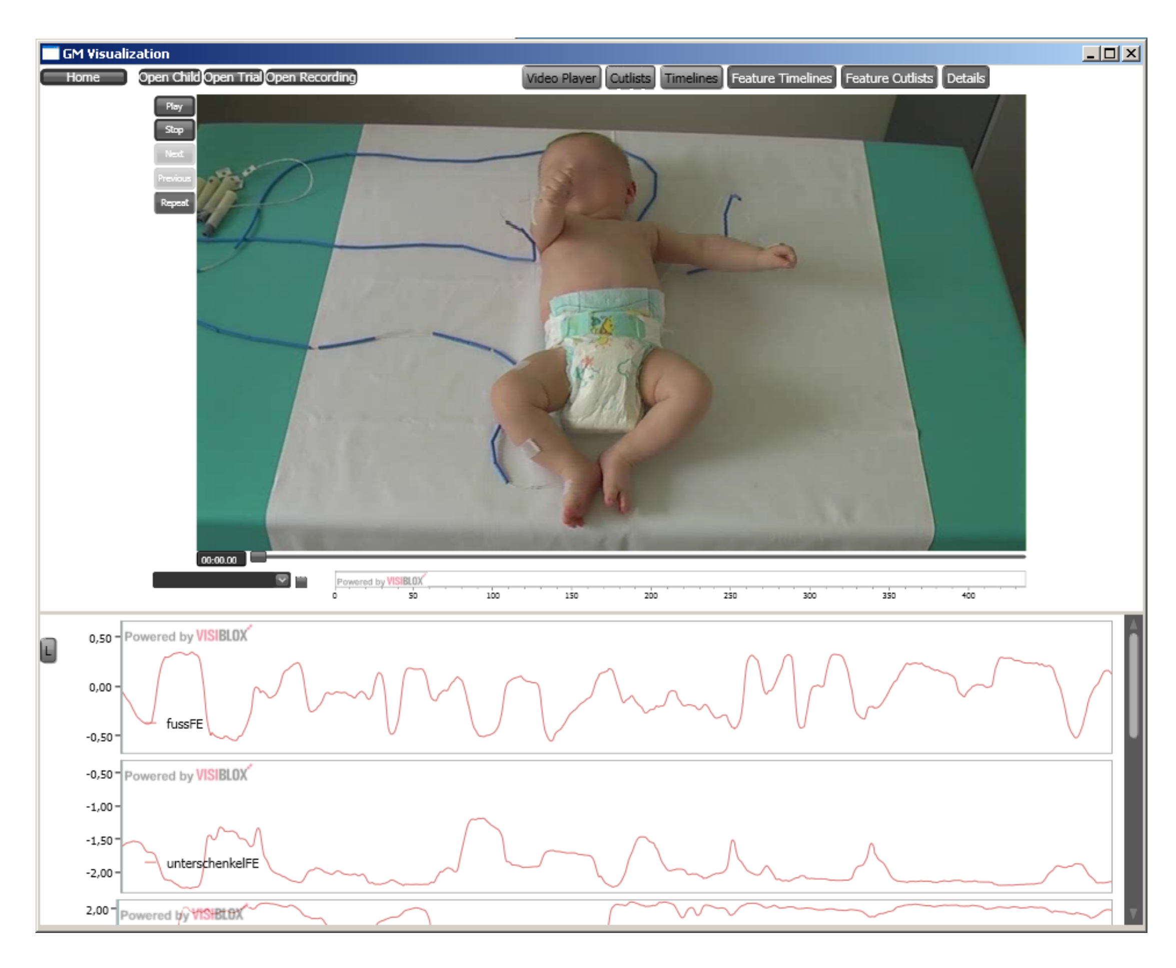


Abbildung 21: Ausgeblendete Detailbereich, Merkmals-Segmente und Merkmals-Zeitreihen.

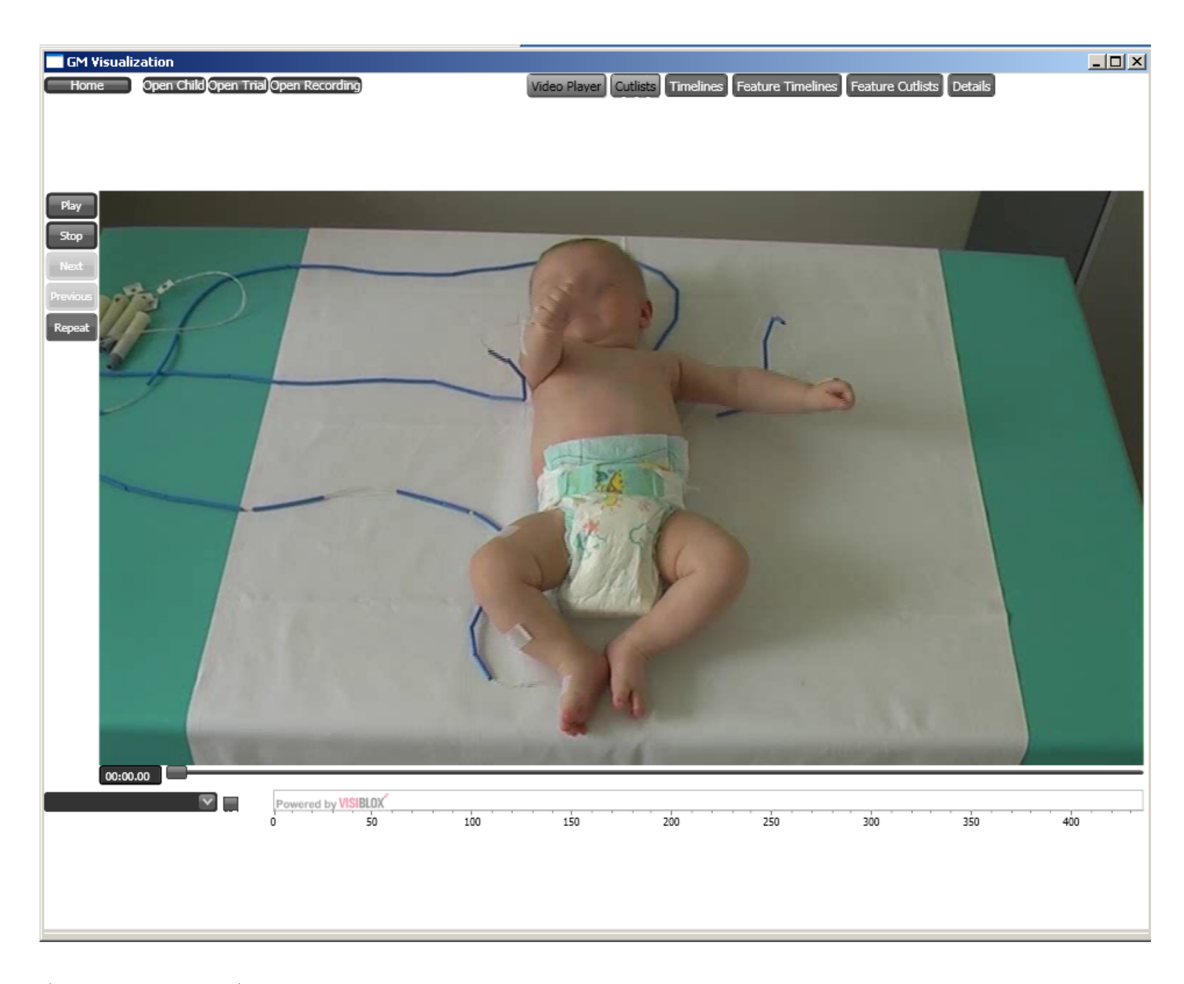


Abbildung 22: Ausgeblendete Detailbereich, Merkmals-Segmente, Merkmals-Zeitreihen und Zeitreihen.

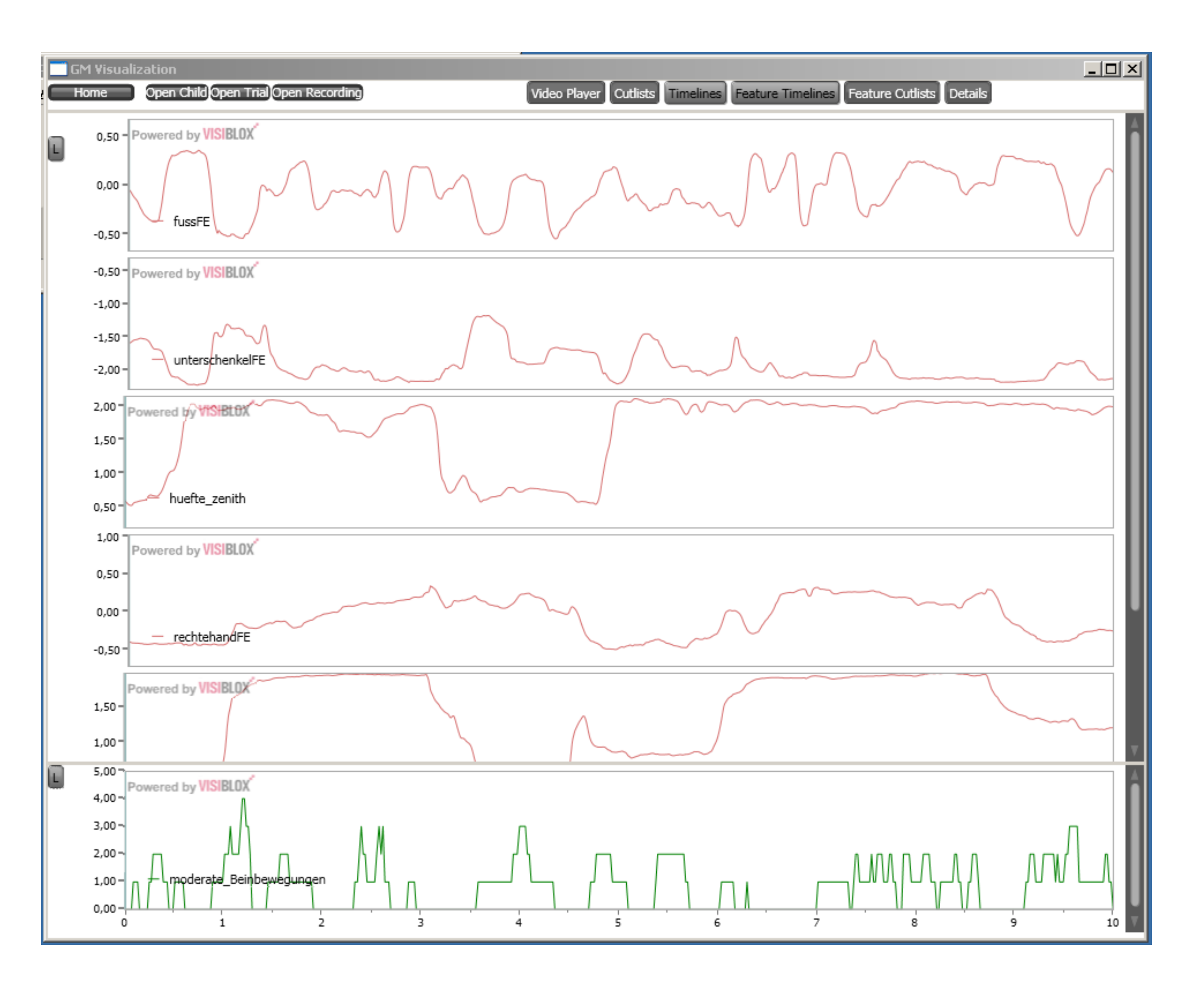


Abbildung 23: Ausgeblendete Detailbereich, Merkmals-Segmente, Segmente und Videoplayer.

3 Konfiguration

Im Folgenden wird beschrieben wie die Anwendung konfiguriert wird.

Die Anwendung wird bei der Auslieferung vorkonfiguriert und sollte so belassen werden.

Die Anwendung wird durch eine Konfigurationsdatei konfiguriert. Der Speicherort dieser Datei ist in der Anwendungskonfiguration *GMVisualization2.exe.config* unter dem Namen „Configuration_File” festgelegt. Listing 1 zeigt die Konfigurationsdatei der vorgestellten Anwendung.

visualization Das ist das Wurzelelement.

- Das Attribut *debug* schaltet den Debug-Modus ein oder aus. In diesem Modus greift die Anwendung nicht auf die Datenbank zu, sondern arbeitet mit Beispieldaten.
- Dass Attribut *defaultRecording* ermöglicht die Angabe einer Teilaufnahme die standardmäßig geladen wird, so dass der Benutzer nur das Kind und die Aufnahme wählen muss, wenn er einen Fall lädt.

features Das Element kann beliebig viele *features* enthalten.

- Durch das Attribut *directory* wird das Verzeichnis festgelegt, worin die Anwendung nach Merkmalsmodulen sucht.

feature Das Element kann beliebigen XML Text als Inhalt haben. er wird verwendet, um das Merkmals-Modul zu konfigurieren.

- Attribut *name* Definiert den Namen des Merkmals in der Datenbank. Er wird verwendet, um in der Datenbank nach dem Merkmal zu suchen.
- Attribut *classificator* gibt den Namen des Klassifikators an, der für das Merkmal verwendet werden soll.
- Attribut *id* gibt den Identifizierer des Merkmals an, er muss mit dem Identifizierer übereinstimmen, der im Merkmals-Modul definiert ist.

feature/configuration Das ist das Wurzelelement der Konfiguration des hier verwendeten Merkmals.

feature/timeline Konfiguriert eine Merkmals-Zeitreihe. Es können beliebig viele Zeitreihen angegeben werden. Die Reihenfolge der hier angegebenen Zeitreihen wird bei der Anzeige beibehalten.

- Attribut *name* gibt den Namen der Zeitreihe in der Datenbank an. Er wird verwendet, um die Zeitreihe aus der Datenbank zu laden.

feature/cutlist Konfiguriert eine Merkmals-Segmentliste. Es können beliebig viele Segmentlisten angegeben werden. Die Reihenfolge wird wie bei den Merkmals-Zeitreihen bei der Anzeige beibehalten.

Listing 1: Konfigurationsdatei.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<visualization xmlns="http://www.uni-heidelberg.de/GMVisualization"
    debug="true" defaultRecording="trial_arm_bein">
    <features directory="C:\BerensteinEugen\Diplomarbeit\VS Projekte\
        Projects\GMVisualization2\Features">
        <feature
            name="moderate_Bewegungen_rechtes_Bein"
                classificator="dummyclassificator" id="feature2ri"
                >
            <configuration xmlns="http://www.uni-heidelberg.de
                /GMVisualization/Feature">
                <timeline name="moderate_Beinbewegungen"/>
                <cutlist name="CutSegment"/>
                <cutlist name="FidgetySegment_arm"/>
                <range minimum="0" maximum="10"/>
                <graph>
                    <dropouts visible="false"/>
                    <unknowns visible="false"/>
                </graph>
            </configuration>
        </feature>
    </features>
    <timelines>
        <timeline name="fussFE"/>
        <timeline name="unterschenkelFE"/>
        <timeline name="huefte_zenith"/>
        <timeline name="rechtehandFE"/>
        <timeline name="rechterunterarmFE"/>
        <timeline name="schulter_zenith"/>
        <range begin="0" end="30"/>
    </timelines>
    <cutlists default="CutSegment">
        <cutlist name="muster"/>
    </cutlists>
    <unitconverter accuracy="maximum" units="seconds"/>
    <classificators directory="C:\BerensteinEugen\Diplomarbeit\VS
        Projekte\Projects\GMVisualization2\Classificators">
        <classificator name="dummyclassificator">
            <featureRef id="feature2ri"/>
        </classificator>
    </classificators>
    <video root="X:\"/>
</visualization>
```

- Attribut *name* gibt den Namen der Segmentliste in der Datenbank an. Er wird genutzt, um die Liste in der Datenbank zu finden.

feature/range Beschreibt die Breite des Ausschnitts aus den Zeitreihen, der dargestellt werden soll. Wird das Element weggelassen, wird die gesamte Zeitreihe auf einmal dargestellt.

- Die Attribute *begin* und *end* geben den Anfang und das Ende des Ausschnitts an. Die Einheit dieser Angaben entspricht den konfigurierten Einheiten der Anwendung (siehe das *unitconverter* Element).

feature/graph Das Element enthält Einstellungen für den Merkmalsgraphen.

feature/dropouts Konfiguriert wie Fälle zu behandeln sind, die aus der Studie ausgeschieden sind.

- Das Attribut *visible* gibt an, ob sie im Merkmalsgraphen angezeigt werden sollen.

feature/unknowns Konfiguriert wie Fälle zu behandeln sind, deren Outcome unbekannt ist.

- Das Attribut *visible* gibt an, ob sie im Merkmalsgraphen angezeigt werden sollen.

timelines Enthält Konfigurationseinstellungen für Zeitreihen. Es kann beliebig viele *timeline* Elemente enthalten.

timeline Analog zu *feature/timeline.*

cutlists Enthält Konfigurationseinstellungen für Segmentlisten. Es kann beliebig viele *cutlist* Elemente enthalten.

- Das Attribut *default* gibt die Standardsegmentliste an.

cutlist Analog zu *feature/cutlist.*

unitconverter Konfiguriert die zu verwendenden Einheiten für Zeitreihen und Segmentlisten.

- Das Attribut*accuracy* gibt die Genauigkeit in Stellen nach dem Komma an, die bei der Umrechnung von Einheiten verwendet werden soll. „maximum" bedeutet, dass die maximale Genauigkeit von 64 Bit Gleitkommazahlen verwendet wird.
- Das Attribut *units* gibt die Einheiten an, die für die Darstellung von Zeitreihen und Segmentlisten verwendet werden. Es können Sekunden, Millisekunden oder Samples (Abtastwerte) sein.

classificators Enthält Konfigurationseinstellungen für Klassifikatoren.

- Das Attribut *directory* gibt das Verzeichnis an, wo die Anwendung nach Klassifikator-Implementierungen suchen soll.

classificator Konfiguriert den globalen Klassifikator. Er kann beliebig viele *featureRef* Elemente enthalten.

- Sein Attribut *name* gibt den Namen des Klassifikators an, der als globaler Klassifikator verwendet werden soll.

featureRef Konfiguriert Merkmale, die als Eingabedaten für den Klassifikator dienen sollen.

- Das Attribut *id* gibt den Identifizierer des Merkmals an.

video Konfiguriert die Videowiedergabe.

- Das Attribut *root* gibt das Wurzelverzeichnis im Dateisystem an, in dem Videoaufnahmen abgelegt sind.

B. Benutzerhandbuch Konfigurationskomponente

1 Einleitung

Die Konfigurationskomponente erleichtert dem Forscher die Konfiguration der Anwendung zur Extrahierung von Merkmalen. Sie bietet semantische Unterstützung und bessere Übersichtlichkeit bei der Erstellung eines Konfigurationscripts.

1.1 Überblick

Das Anwendungsfenster enthält fünf Bereiche. Abbildung 1 zeigt das Anwendungsfenster (die Darstellung des Konfigurationscripts ist aus Übersichtsgründen ausgeblendet).

Auf der linken Seite befindet sich die *Toolbox*. In diesem Fenster befinden sich die Werkzeuge zum Erstellen des Scripts. Sie sind in klappbaren Gruppen organisiert.

Auf der unteren Seite sehen Sie die *Fehlerliste*. In diesem Fenster werden Fehler und Warnungen der Anwendung ausgegeben.

Auf der rechten Seite oben befindet sich der *DSL Explorer*. In diesem Fenster wird das Script in einer Baumstruktur dargestellt. Hier kann das Script auch bearbeitet werden, indem Elemente gelöscht oder hinzugefügt werden.

Unter dem DSL Explorer ist das *Eigenschaftenfenster*. In diesem Fenster werden Eigenschaften von im Script markierten Objekten angezeigt und können bearbeitet werden.

In der Mitte des Anwendungsfensters ist der Bereich, wo das Konfigurationscript angezeigt wird und bearbeitet werden kann.

Zusätzlich befinden sich oben einige Werkzeuge zum Öffnen und Speichern von Dateien, Rückgängig machen und Wiederholen von Aktionen, sowie zum Kopieren und Einfügen.

Jeder dieser Bereiche kann beliebig positioniert und in der Größe verändert werden.

2 Arbeiten mit der Konfigurationskomponente

In den Folgenden Abschnitten wird Beschrieben wie Sie die Schritte in einem typischen Arbeitsablauf mit der Konfigurationskomponente durchführen außerdem werden die verschiedenen Elemente und Werkzeuge der Anwendung erklärt.

2.1 Laden und Anlegen

Um ein neues Script anzulegen, navigieren Sie zum Menü *Datei* → *Neu*→ *Datei...* und wählen Sie die Vorlage *GMAlgorithmConfigurator*. Die Anwendung legt zwei Dateien an und öffnet sie. Die Datei mit der Endung *.gmc* enthält das eigentliche Script. Die Datei mit der Endung *.gmc.diagramm* enthält Informationen über das Layout des Scripts im Editor.

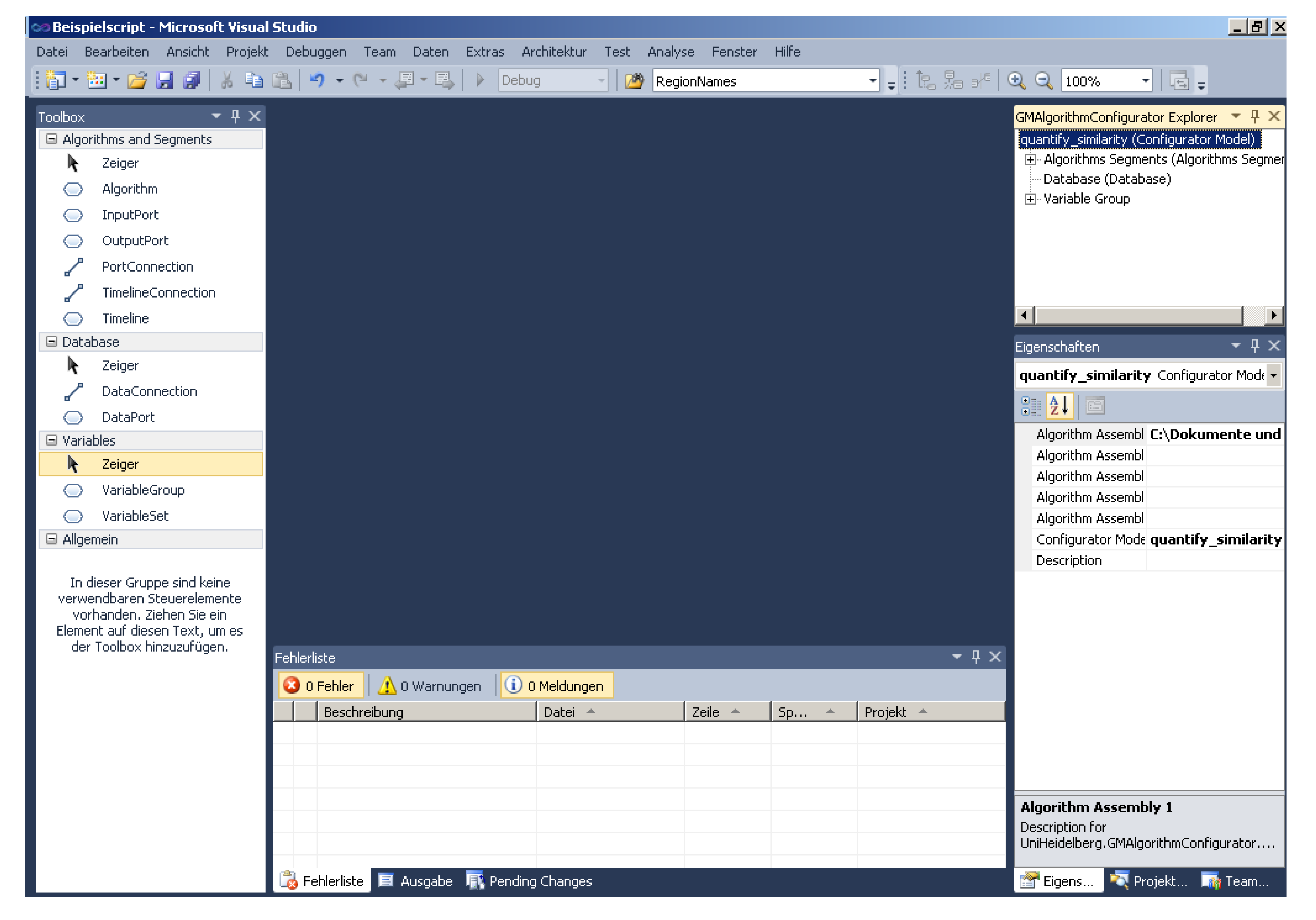


Abbildung 1: Programmfenster ohne geladenes Script.

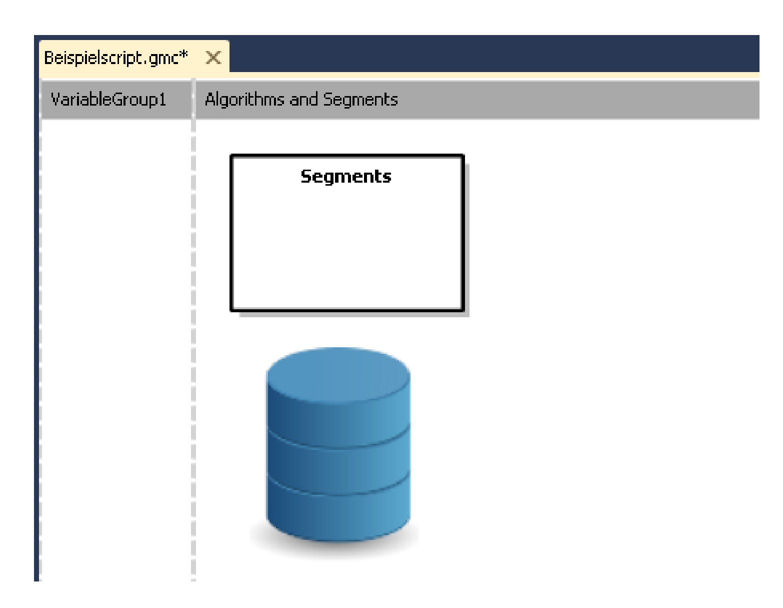


Abbildung 2: Leeres Script.

Der Verlust der .gmc.diagramm Datei führt nur zum Verlust der Anordnung der Elemente, alle Elemente und Verbindungen bleiben erhalten. In diesem Fall legt die Anwendung automatisch eine neue Layoutdatei an und ordnet die Elemente zufällig an.

Klicken Sie auf das Symbol *Speichern*, und wählen Sie einen Speicherort, um das Script zu speichern.

Um ein Script in der Anwendung zu öffnen, navigieren Sie zum Menü *Datei* → *Öffnen* → *Datei...* und öffnen Sie die Datei mit der Endung *.gmc*. Das Script wird geladen und angezeigt. Falls beim Laden ein Fehler auftritt, können Sie den Fehler in der Fehlerliste sehen.

2.2 Script Bearbeiten

Sie beginnen mit einem leeren Konfigurationscript. Die Anwendung legt dabei automatisch einige Elemente an. Abbildung 2 zeigt das Beispielscript nach dem Laden.

Der Editorbereich ist in zwei „Schwimmbahnen" geteilt. *VariableGroup1* ist eine Variablengruppe. *Algorithms and Segments* ist der Bereich, der die Hauptelemente des Konfigurationscripts enthält. In der „Schwimmbahn" *Algorithms and Segments* befindet sich ein Rechteck mit dem Namen *Segments*. Dieses Element enthält die Konfiguration der Segmente. Außerdem ist darunter ein blauer Zylinder. Dieses Element repräsentiert die Datenbank.

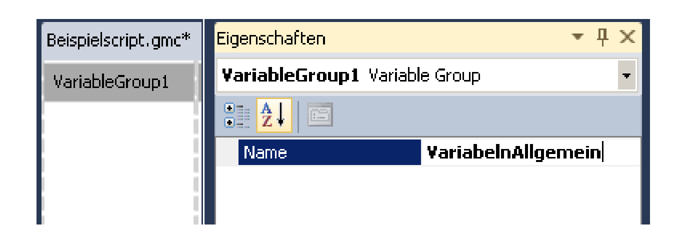


Abbildung 3: Variablengruppe umbenennen.

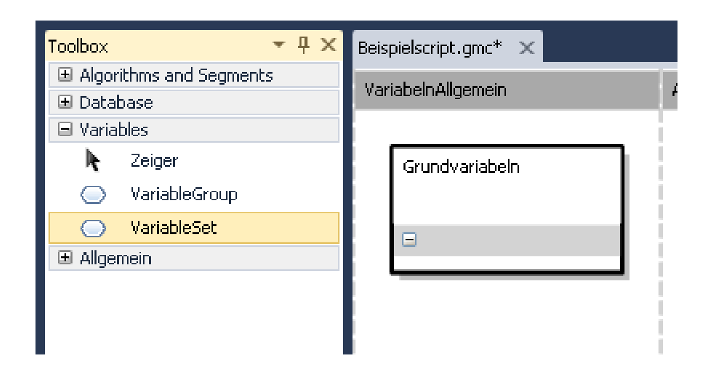


Abbildung 4: Variablenmenge hinzufügen.

2.2.1 Variablen

Wir benennen VariableGroup1 in „VariabelnAllgemein" um. Wir selektieren dazu VariableGroup1 und ändern den Namen im Eigenschaftenfenster (Abbildung 3). Sie können weitere Variablengruppen hinzufügen, indem Sie sie aus der Toolbox auf einen leeren Bereich des Editors ziehen (außerhalb der „Schwimmbahnen").

Wir fügen eine *VariableSet* zur Variablengruppe hinzu. Dazu ziehen wir eine VariableSet aus der Toolbox auf die Variablengruppe und benennen sie in „Grundvariabeln" um (Abbildung 4). Eine VariableSet ist eine Variablenmenge, sie gruppiert einzelne Variablen.

Als nächstes fügen wir der Variablenmenge eine *Variable* hinzu. Dazu klicken wir mit der rechten Maustaste auf die Variablenmenge und navigieren zu *Hinzufügen* → *Variable*. Wir benennen die Variable in Anzahl um und tragen bei *Value* den Wert 100 ein (Abbildung 5). Der Name wird automatisch mit dem Präfix *var_* ergänzt. Ein Variable kann benutzt werden, um Algorithmenparameter zu konfigurieren. Das ermöglicht ein Austauschen der Werte bei der Ausführung des Scripts ohne das Script ändern zu müssen.

2.2.2 Verbinden des Scripts mit Algorithmen

Um mit dem Editor arbeiten zu können, müssen sie ihm zuerst den Speicherort der Assemblies mitteilen, die Algorithmen enthalten, mit denen Sie arbeiten möchten.

Klicken Sie dazu auf eine freie Stelle im Editorfenster außerhalb der „Schwimmbahnen" (verschieben Sie dazu die Ansicht weiter nach rechts). Suchen Sie das Eigenschaftenfenster

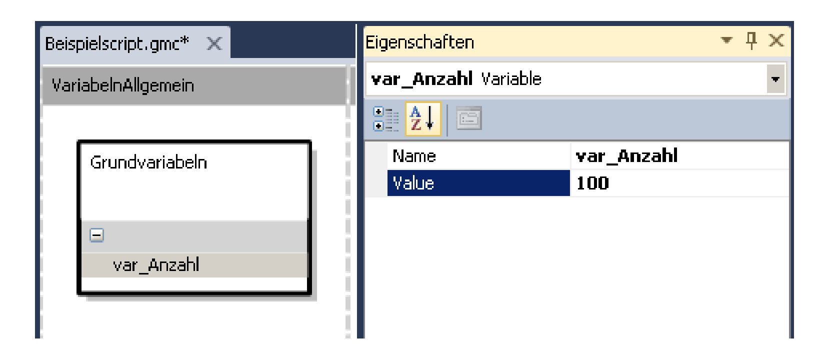


Abbildung 5: Variable hinzufügen.

(Wenn Sie es nicht sehen können, öffnen Sie es durch einen Druck auf die Taste *F4*. Unter der Überschrift *Assemblies* sind dort Eintragungen *Algorithm Assembly 1 .. 5*. Markieren Sie eine Eintragung und klicken Sie auf den Button mit den drei Punkten rechts davon. Suchen Sie nun die Assembly und öffnen Sie sie.

Der Editor analysiert die von ihnen gewählte Assembly und lädt Informationen über die Algorithmen. Wenn der Editor beim Laden einer Assembly auf fehlerhafte Informationen stößt, gibt er eine Warnung in der Fehlerliste aus.

Die Informationen über die Algorithmen werden nicht mit dem Script gespeichert, sondern bei jedem Laden des Scripts erneut eingelesen. Wenn die Anwendung eine oder mehrere Algorithmen nicht finden kann, werden entsprechende Fehlermeldungen in der Fehlerliste angezeigt und das Script kann nicht korrekt bearbeitet werden.

2.2.3 Algorithmen

Nachdem die Informationen über Algorithmen geladen wurden, können wir einen Algorithmus hinzufügen. Ziehen Sie dazu einen *Algorithm* aus der Toolbox auf die *Algorithms and Segments* „Schwimmbahn". Daraufhin erscheint ein Fenster dass die Auswahl eines Namespace und eines Algorithmus ermöglicht (Abbildung 6).

Abbildung 7 zeigt die Darstellung des Algorithmus im Editor. Er enthält *Eingabeports* (grün), *Ausgabeports* (rot) und *Datenports* (gelb).
Eingabeports dienen dazu dem Algorithmus Daten zu übergeben. Ausgabeports sind Daten, die der Algorithmus ausgibt. Datenports repräsentieren Daten, die nach der Ausführung des Algorithmus in der Datenbank gespeichert werden sollen.
Der Algorithmus enthält außerdem *Properties*, das sind die Parameter des Algorithmus, sowie *Extremities*, hier werden Extremitäten definiert, mit denen der Algorithmus arbeiten soll.

Als Nächstes definieren wir Segmente für den Algorithmus. Zuerst selektieren wir Segments und tragen in den Eigenschaften unter Output „seg" ein. Danach selektieren wir den Algorithmus und wählen in den Eigenschaften unter Segment „seg" aus (Abbildung 8).

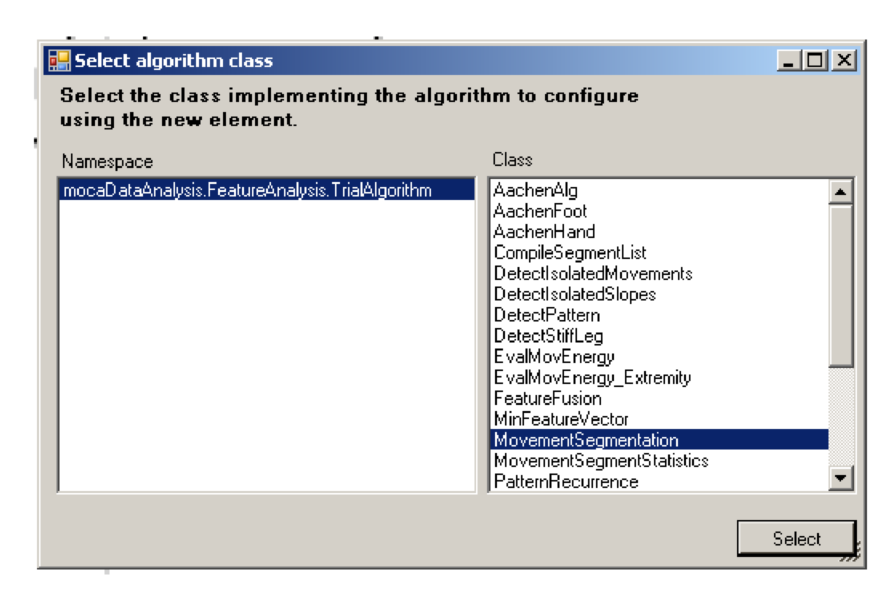


Abbildung 6: Algorithmus auswählen.

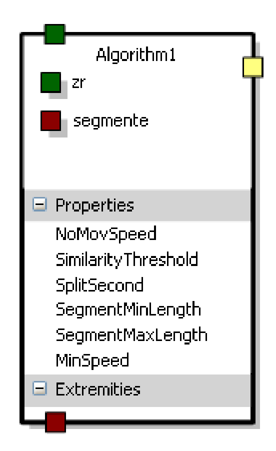
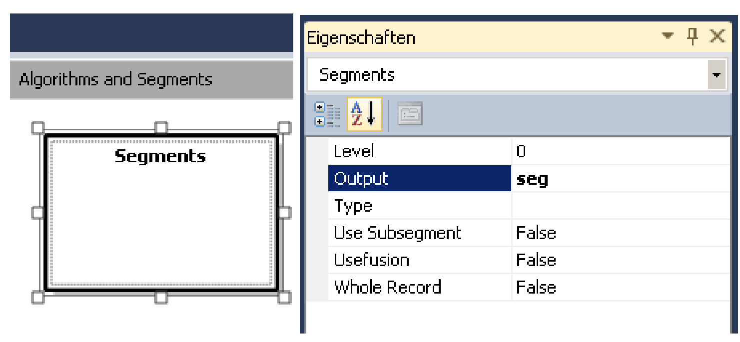
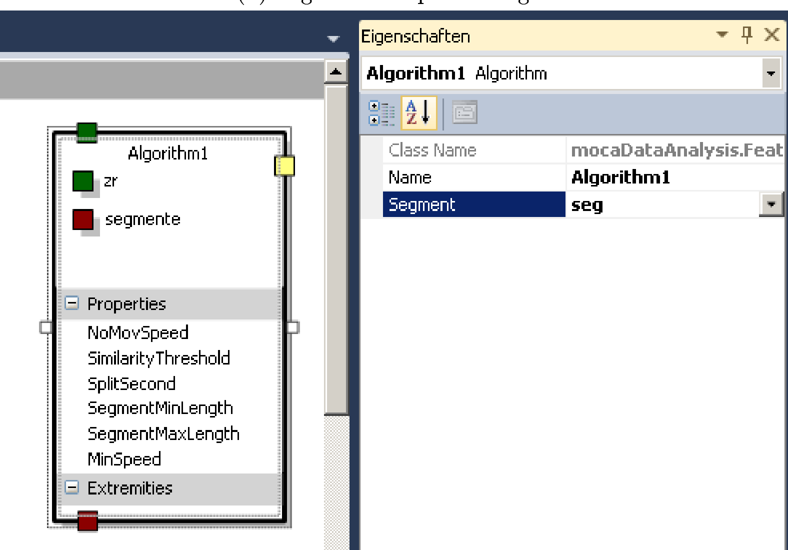


Abbildung 7: Algorithmus.



(a) Segment Output festlegen.



(b) Segmente beim Algorithmus eintragen.

Abbildung 8: Segmente definieren.

Abbildung 9: Properties hinzufügen.

Jetzt fügen wir dem Algorithmus ein *Extremity* hinzu. Dazu klicken wir mit der rechten Maustaste auf die Überschrift *Extremities* im Algorithmus und wählen „neues Extremity hinzuzufügen". Dann nennen wir die Extremity „Arm".

Jetzt versuchen wir eine *Property* hinzuzufügen. Dazu gehen wir analog zu Extremities vor, doch es existiert kein Menüpunkt zum Anlegen von Properties. Das liegt daran dass die Anwendung bereits alle definierten Properties automatisch angelegt hat. Sie können keine Properties für einen Algorithmus anlegen, die nicht definiert sind.
Wir müssen zuerst einige Properties löschen. Wenn wir jetzt versuchen eine Property anzulegen, erscheint ein Fenster, wo wir die Property auswählen können (Abbildung 9). Wenn das Fenster nicht erscheint, dann stand nur eine Property zur Auswahl, diese wird automatisch hinzugefügt.

Um den Wert einer Property festzulegen, selektieren wir zuerst die Property. Im Eigenschaftenfester unter *Value* können wir entweder den Wert direkt eintragen oder eine zuvor definierte Variable wählen. Abbildung 10 veranschaulicht das.

Übrigens wird im Eigenschaftenfenster unter Type *der Datentyp der Property angezeigt. Sie sollten prüfen, ob der eingegebene Wert diesem Typ entspricht.*

2.2.4 Ports

Sie können einem Algorithmus einen Port hinzufügen, indem Sie den Port aus der Toolbox auf den Algorithmus ziehen. Die Toolbox enthält InputPort, OutputPort und DataPort, sie alle werden auf die gleiche Weise hinzugefügt. Sie können nur einen Port erstellen, wenn es noch nicht hinzugefügte Ports gibt oder wenn einer oder mehrere Ports mehrfach hinzugefügt werden dürfen. Ob ein Port mehrfach hinzugefügt werden darf erfahren Sie, wenn Sie den Port selektieren und im Eigenschaftenfenster den Wert von *Allow Multiple* betrachten. Steht dort *true*, darf der Port mehrfach hinzugefügt werden.

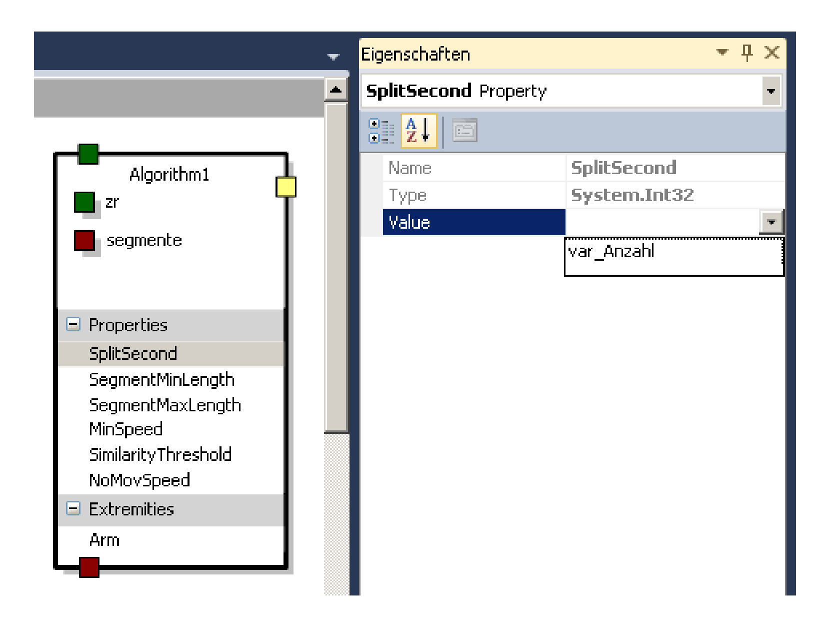


Abbildung 10: Wert der Property festlegen.

Wenn mehrere Ports zur Auswahl stehen wird ein Auswahlfenster angezeigt (Abbildung 11) ansonsten wird analog zu den Properties der einzig verfügbare Port hinzugefügt.

Sie können Ports auch löschen. Dann werden Verbindungen, die an diesen Ports enden oder von ihnen ausgehen automatisch mitgelöscht. Einige Ports können nicht gelöscht werden. Diese Ports sind an andere Ports gebunden und werden mit ihnen zusammen automatisch angelegt und gelöscht. Ein Port der andere Ports bindet hat den Wert *true* bei seiner Eigenschaft *Has Output*. Ein Port, der an einen anderen gebunden ist hat den Wert *true* bei seiner Eigenschaft *Is Attached*.

2.2.5 Verbindungen

Verbindungen können zwischen Ausgabeports und Eingabeports sowie zwischen Ports und anderen Elementen existieren

Wir wollen einen Eingabeport mit einer Zeitreihe verbinden. Dazu legen wir zuerst die Zeitreihe an, indem wir eine *Timeline* aus der Toolbox auf die *Algorithms and Segments* „Schwimmbahn" ziehen. Dann klicken wir in der Toolbox auf das Werkzeug *TimelineConnection*, klicken auf die Timeline und dann auf den Eingabeport, den wir verbinden können. Abbildung 12 zeigt das Resultat.
Sie können die Elemente nur verbinden, wenn der Eingabeport einen Datentyp hat, der eine Zeitreihe akzeptiert. Den Datentyp des Ports sehen Sie im Eigenschaftenfenster unter *Type*.

Nun verbinden wir die Ports einiger Algorithmen miteinander. Dazu klicken wir auf das *PortConnection* Werkzeug in der Toolbox, klicken auf einen Ausgabeport und dann auf

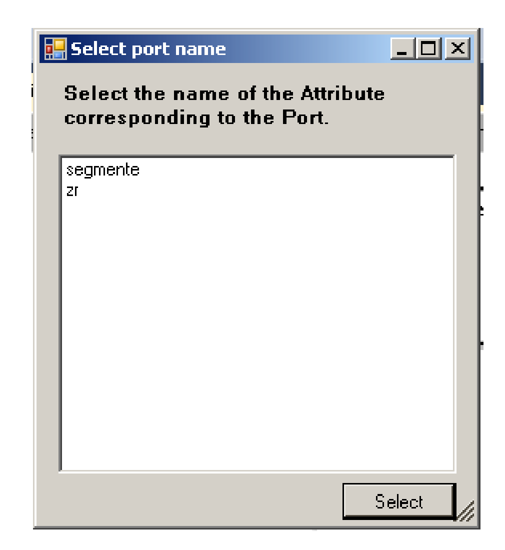


Abbildung 11: Port auswählen.

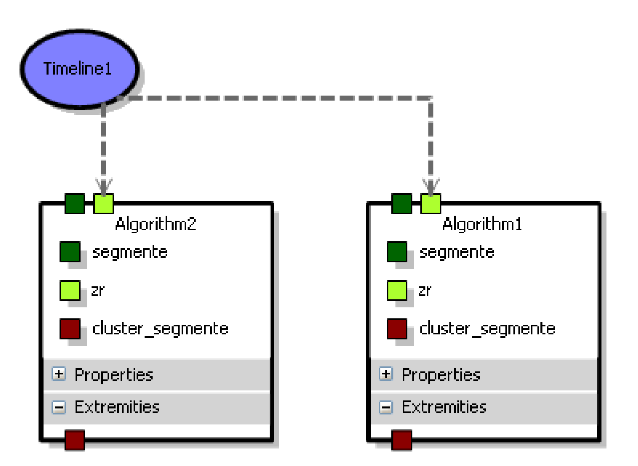


Abbildung 12: Verbindung zwischen Zeitreihe und Eingabeport.

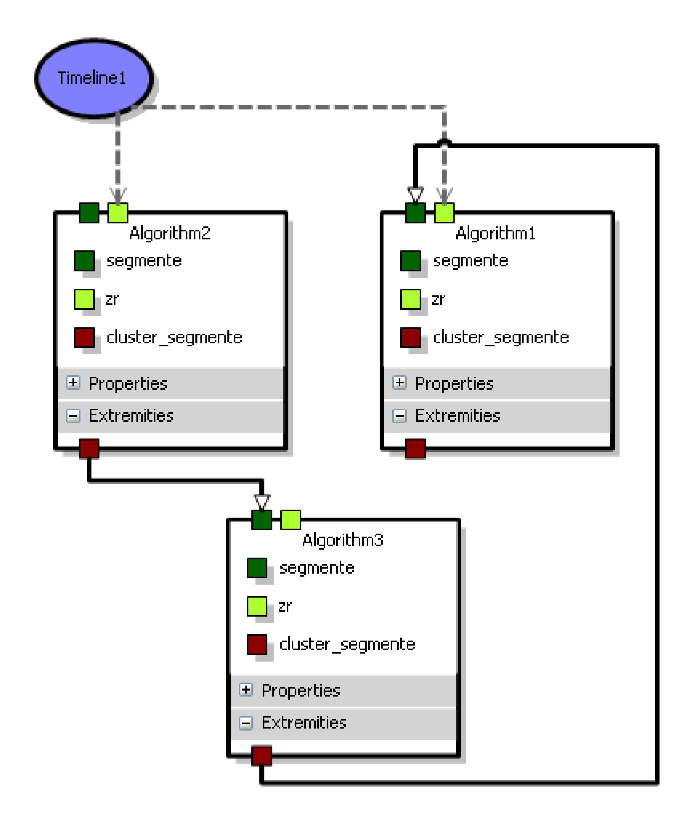


Abbildung 13: Verbindung zwischen Ports von Algorithmen.

einen Eingabeport eines anderen Algorithmus. Abbildung 13 zeigt zwei miteinander verbundene Algorithmen.
Sie können die Ports nur verbinden, wenn ihre Datentypen übereinstimmen. Ferner können Sie zwei Ports nicht verbinden, wenn die Verbindung zu einem Zyklus in den Verbindungen führen würde. Zum Beispiel können Algorithm1 und Algorithm2 in Abbildung 13 nicht verbunden werden, solange die Verbindung von Algorithm2 über Algorithm3 zu Algorithm1 besteht.

Als letztes verbinden wir noch den DataPort des Algorithmus mit der Datenbank. Dazu klicken wir auf das *DataConnection* Werkzeug in der Toolbox, klicken auf einen DataPort und dann auf den Zylinder, der die Datenbank repräsentiert. Abbildung 14 zeigt das Resultat.

3 Konfiguration

Die Konfigurationskomponente wird selbst durch eine Konfigurationsdatei konfiguriert. Diese befindet sich in einem Verzeichnis, das als allgemeines Repository für programms-

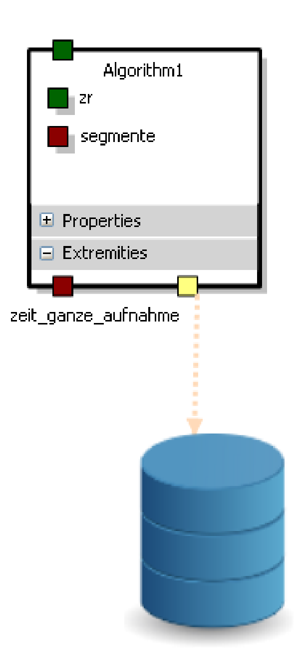


Abbildung 14: Verbindung mit der Datenbank.

pezifische Daten verwendet wird, die von allen Benutzern verwendet werden[1]. Auf einer deutschen Windows XP Installation ist dieses Verzeichnis *C:\Dokumente und Einstellungen\All Users\Anwendungsdaten*. Der Name der Datei lautet *GMConfiguration.xml*.

Listing 1: Inhalt der Konfigurationsdatei.

```
<?xml version="1.0" encoding="utf-8" ?>
<editor    xmlns="http://www.uni-heidelberg.de/GMConfiguration">
    <converter    directory="C:\BerensteinEugen\Diplomarbeit\VS Projekte\Projects\Converters"/>
</editor>
```

Listing 1 zeigt den Inhalt der Konfigurationsdatei. Bisher enthält sie nur ein Element: *converter*. Sein Attribut *directory* enthält das Verzeichnis, wo die Anwendung nach Versionskonvertern sucht.

[1] Beschreibung aus Microsoft Dokumentation zu *Environment.SpecialFolder.CommonApplicationData* im .NET Framework 4.

C. Installationsanleitung Visualisierungskomponente

1 Installationspaket

Das Installationspaket enthält drei Verzeichnisse: *Application*, *Classificators* und *Features*.

Application enthält die ausführbare Datei *GMVisualization2.exe*, die die Anwendung startet. Außerdem Klassenbibliotheken, von denen das Programm abhängig ist und die Anwendungskonfigurationsdatei *GMVisualization2.exe.config*. Im Unterverzeichnis *Configuration* befindet sich die Konfigurationsdatei *configuration.xml*.

Classificators enthält Klassenbibliotheken, die Klassifikatoren implementieren und Klassenbibliotheken, von denen sie abhängig sind.

Features enthält Klassenbibliotheken, die Merkmals-Module implementieren und Klassenbibliotheken, von denen sie abhängig sind.

2 Voraussetzungen

Voraussetzungen für den Betrieb der Anwendung sind ein installiertes *.NET Framework 4* und die Möglichkeit eine Verbindung zur Datenbank des Forschungsprojekts herzustellen.

Auf einem Computer, auf dem das Programm mocaPlayer läuft, müsste auch die Visualisierungskomponente funktionieren.

3 Installation

Kopieren Sie den Inhalt des Installationspakets in ein gemeinsames Verzeichnis auf dem Computer.

Passen Sie die Anwendungskonfiguration an.
Öffnen Sie dazu die Datei *GMVisualization2.exe.config* mit einem Texteditor. Suchen Sie den folgenden Abschnitt unter *applicationSettings* und tragen Sie die Benutzernamen der Benutzer an diesem Computer ein.

```
<setting name="users" serializeAs="Xml">
    <value>
        <ArrayOfString          xmlns:xsi="http://www.w3.org/2001/
            XMLSchema-instance"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <string>user1</string>
            <string>user2</string>
        </ArrayOfString>
    </value>
</setting>
```

Wenn sie die Konfigurationsdatei an einen anderen Ort als im Unterverzeichnis *Configuration* im Verzeichnis, wo sich die ausführbare Datei der Anwendung befindet, verschoben haben suchen Sie den folgenden Abschnitt unter *userSettings* und tragen Sie das neue Verzeichnis ein.

```
<setting name="Configuration_File" serializeAs="String">
    <value>Configuration\configuration.xml</value>
</setting>
```

Suchen Sie den folgenden Abschnitt unter *log4net* und tragen Sie dort den Pfad ein, wo die Logdatei angelegt werden soll.

```
<appender name="FileWriter" type="log4net.Appender.
    RollingFileAppender">
    <file value="C:/WINNT/Profiles/All Users/Anwendungsdaten/
        GMVisualization/Logging/logging.txt"/>
...
```

Passen sie die Konfiguration an.
Öffnen Sie die Konfigurationsdatei *configuration.xml* mit einem Texteditor.
Suchen Sie das Tag *features* und tragen Sie beim Attributwert für *directory* den absoluten Pfad zum Verzeichnis ein, in dem Merkmals-Module abgelegt werden.
Suchen Sie das Tag *classificators* und tragen Sie beim Attributwert für *directory* den absoluten Pfad zum Verzeichnis ein, in dem Klassifikatoren abgelegt werden.

D. Installationsanleitung Konfigurationskomponente

1 Einleitung

Der Editor kann auf zwei Arten installiert werden. Als Teil einer bestehenden Visual Studio 2010 Installation oder als eigenständiges Programm (Standalone).

2 Inhalt des Pakets

Das Installationspaket enthält unterschiedliche Unterverzeichnisse. Im Folgenden wird ihr Zweck und Inhalt erklärt.

gac Enthält .net Assemblies, die in den Global Assembly Cache installiert werden müssen.

dev Enthält .net Assemblies und zugehörige Dokumentationsdateien für Entwickler. *ConverterInterface.dll* und **.xml* werden zur Entwicklung von Versionskonvertern zwischen verschiedenen Diagrammversionen benötigt. *UniHeidelberg.ScriptAttributes.dll* und **.xml* sowie *mocaDataAnalysis_Interfaces.dll* werden zur Entwicklung von Algorithmen benötigt, die vom Editor geladen werden können.

license Enthält Lizenzen von Drittanbieterkomponenten.

setup Enthält ausführbare Dateien und Komponenten, die für die Installation benötigt werden.

xml Enthält XML Schemata für die Validierung der Konfigurationsdatei und der Diagramme.

3 Installation in Visual Studio 2010

Der Editor ist im Grunde eine Erweiterung von Visual Studio 2010. Bei dieser Art von Installation integriert er sich in ihre bestehende Visual Studio 2010 Installation.

3.1 Voraussetzungen

Voraussetzung ist ein installiertes Visual Studio 2010.

3.2 Installationsschritte

3.2.1 Installation der Assemblies

Installieren Sie die Assemblies aus dem Verzeichnis *gac* in den Global Assembly Cache (GAC).

1. Gehen Sie dazu auf *Start → Alle Programme → Microsoft Visual Studio 2010 →Visual Studio Tools → Visual Studio Eingabeaufforderung (2010)*, um die Visual Studio Eingabeaufforderung zu starten.
2. Geben Sie in der Eingabeaufforderung folgendes ein:

   ```
   gacutil -i [Pfad][Datei]
   ```

   Ersetzen Sie dabei *[Pfad]* durch den vollständigen Pfad zu der jeweiligen Datei und *[Datei]* durch den Dateinamen mit Endung. Wiederholen Sie diesen Schritt für alle oben genannten Dateien.

3.2.2 Installation des Editors

1. Starten Sie dazu die Datei *GMParameterConfigurator.DslPackage.vsix* aus dem Verzeichnis *setup*. Im sich jetzt öffnenden Fenster bestätigen Sie die Installation in Visual Studio 2010.

3.2.3 Installation des XML Schemas (Optional)

Um eine Warnung im Fehlerfenster des Editors zu vermeiden, können Sie das XML Schema für Diagramme installieren.

1. Kopieren Sie dazu die Datei *GMAlgorithmConfiguratorSchema.xsd* aus dem Verzeichnis *xml* in das Verzeichnis *[Installationspfad von Visual Studio 2010]\Xml\Schemas.* Der Installationspfad lautet normalerweise *C:\Programme\Microsoft Visual Studio 10.0*

4 Standalone Installation

Der Editor kann in einer speziellen Visual Studio Umgebung installiert werden, die als eigenständiges Programm ausgeführt wird, der Visual Studio 2010 Isolated Shell.

4.1 Voraussetzungen

Voraussetzung ist bei Windows XP ein installiertes Service Pack 3.

4.2 Installationsschritte

4.2.1 Installation von .NET Framework 4.0

1. Öffnen Sie die Datei *dotNetFx40_Full_x86_x64.exe* aus dem Verzeichnis *setup* und folgen Sie den Anweisungen.

4.2.2 Installation von Visual Studio Shell

1. Öffnen Sie die Datei *VSIsoShell.exe* aus dem Verzeichnis *setup* und folgen Sie den Anweisungen.

4.2.3 Installation der Editorumgebung

1. Öffnen Sie die Datei *GMIsoShell.msi* aus dem Verzeichnis *setup* und warten Sie bin die Installation abgeschlossen ist.

4.2.4 Installation des Editors

1. Falls Sie kein Visual Studio 2010 installiert haben, öffnen Sie die Windows Eingabeaufforderung und navigieren Sie zum Pfad, wo sich das Tool *gacutil.exe* befindet (im Verzeichnis *setup/gacutil* des Installationspakets). Andernfalls führen Sie den Schritt 1. aus 3.2.1 durch.
2. Führen Sie jetzt den Schritt 2. aus 3.2.1 durch.
3. Führen Sie jetzt den Schritt 1 aus 3.2.2 durch. Achten Sie darauf, *GMParameterSchell* als Installationsziel auszuwählen.
4. (Optional) Sie können jetzt noch den Schritt 1 aus 3.2.3 durchführen, um eine Warnung des Editors zu vermeiden.

Studiengang Medizinische Informatik

Masterstudiengang Informationsmanagement in der Medizin

....

(Name, Vorname) (Matrikelnummer)

Thema der Diplom-/Masterarbeit: ....

....

....

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.
Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistung von folgenden Personen erhalten:

....

....

....

....

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und ist auch noch nicht veröffentlicht.

....

(Ort, Datum) (Unterschrift)